

CUBIT Mesh Generation Environment Volume 1: Users Manual

Cubit Development Team¹
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

The CUBIT mesh generation environment is a two- and three-dimensional finite element mesh generation tool which is being developed to pursue the goal of robust and unattended mesh generation—effectively automating the generation of quadrilateral and hexahedral elements. It is a solid-modeler based pre-processor that meshes volume and surface solid models for finite element analysis. A combination of techniques including paving, mapping, sweeping, and various other algorithms being developed are available for discretizing the geometry into a finite element mesh. CUBIT also features boundary layer meshing specifically designed for fluid flow problems. Boundary conditions can be applied to the mesh through the geometry and appropriate files for analysis generated. CUBIT is specifically designed to reduce the time required to create all-quadrilateral and all-hexahedral meshes. This manual is designed to serve as a reference and guide to creating finite element models in the CUBIT environment.

This manual documents CUBIT Version 1.11.0.

1. See the next page for the members of the CUBIT Development Team.

▼ Cubit Development Team Membership

Sandia National Laboratories, Albuquerque New Mexico

James R. Hipp	Computational Mechanics & Visualization
Randy R. Lober	Advanced Engineering & Manufacturing Software
Scott A. Mitchell	Applied & Numerical Mathematics
Gregory D. Sjaardema	Solid & Structural Mechanics
Marilyn K. Smith	Technology Programs
Timothy J. Tautges	Computational Mechanics & Visualization
Tammy J. Wilson	Technology Programs

Los Alamos National Laboratories, Los Alamos, New Mexico

William R. Oakes	Technology Modeling and Analysis Group
------------------	--

Brigham Young University, Salt Lake City, Utah

Steve Benzley	Civil Engineering Department
Steve Owen	
Mark Whitely	
David White	

Consultants

Malcolm Panthaki	Consultant, Albuquerque, New Mexico
------------------	-------------------------------------

▼ Table of Contents

▼ Cubit Development Team Membership	4
▼ Table of Contents	5
▼ List of Figures	13
▼ List of Tables	15
 Chapter 1: Getting Started	 17
▼ How to Use This Manual	17
▼ CUBIT Mailing List	18
▼ Problem Reports and Enhancement Requests	18
▼ Executing CUBIT	19
Execution Command Syntax	19
Initialization File	21
User Environment Settings	21
Graphics Customization	21
▼ Command Syntax	22
▼ Features	24
Geometry Creation	24
Algebraic Command Preprocessing	24
Geometry Consolidation	24
Geometry Decomposition	25
Supported Element Types	25
Mesh Creation	25
Boundary Condition Application	25
Graphical Display Capabilities	26
Hardware Platforms	26
▼ Future Releases	26
 Chapter 2: Tutorial	 27
▼ The Tutorial	27
▼ Step 1: Beginning Execution	29
▼ Step 2: Creating the Brick	29
▼ Step 3: Creating the Cylinder	31
▼ Step 4: Adjusting the Graphics Display	31
▼ Step 5: Forming the Hole	32
▼ Step 6: Setting Body Interval Size	33
▼ Step 7: Setting Specific Surface Intervals	33
▼ Step 8: Setting Specific Curve Intervals	34
▼ Step 9: Surface Meshing	35
▼ Step 10: Volume Meshing	35
▼ Congratulations!	37

Chapter 3: Environment	39
▼ Interface Choices	39
Overview	39
Command Line Version	39
Batch Interface	41
▼ Session Control	41
General Execution Commands	41
▼ Journal Files	42
CUBIT Journal File Generation	42
Replaying Journal Files	43
▼ Graphics	44
Graphics Window Control	44
Image Rendering Control	45
Viewing the Model	47
Displaying Entities	50
Drawing Entities 50	
Highlighting Entities 51	
Setting Visibility 51	
Global Settings	52
Individual Geometric Entity Settings	53
Color	53
Entity Labeling	53
Hardcopy Output	55
Video Animations	55
▼ Model Information	56
Model Summary Information	56
Geometry Information	57
Mesh Information	58
Special Entity Information	60
Other Information	61
Message Output Settings 62	
Graphical Display Information 63	
Memory Usage Information 63	
▼ Picking	64
▼ Help Facility	65
Chapter 4: Geometry	67
▼ Geometry Definition	67
Geometric Topology	67
Vertex 67	
Curve 67	
Surface 68	
Volume 68	

Body	68
Group	68
Cellular Topology	68
▼ Geometry Creation	69
Geometry Primitives	70
Brick	70
Cylinder	71
Prism	72
Frustum	73
Pyramid	73
Sphere	74
Torus	74
Sketchpad Geometry	75
Sketch Overview	76
Polygonal Outline	76
Outline Refinement	77
Importing Geometry	77
Importing ACIS Files	77
ACIS Test Harness	77
PRO/Engineer	77
FASTQ	78
▼ Geometry Manipulation	78
Transform Operations	78
Copy	78
Move	79
Scale	80
Rotate	80
Reflect	81
Restore	81
Boolean Operations	82
Intersect	82
Subtract	82
Unite	83
▼ Geometry Decomposition	84
Web Cutting	84
Body-Based Decomposition	85
▼ Geometry Consolidation	85
General Geometry Consolidation	86
Selective Geometry Consolidation	87
▼ Geometry Attributes	88
Entity Names	88
 Chapter 5: Mesh Generation	 89
▼ Mesh Definition	89

Table of Contents

Mesh Hierarchy.....	89
Node	89
Edge	89
Face	90
Brick	90
Mesh Generation.....	90
▼ Mesh Attributes	90
Meshing Schemes	90
Interval Specification	91
Element Types	91
▼ Surface Vertex Types	92
▼ Automated Interval Assignment	92
Scheme Map Interval Assignment Constraints.....	94
Scheme Submap Interval Assignment Constraints.....	95
▼ Curve Meshing	95
Node Density	95
Relative Element Edge Lengths	97
Sizing Function-Based Node Placement.....	97
Meshing the Curve	98
▼ Surface Meshing	98
Scheme Designation	99
Mapping	99
Paving	99
Submapping	100
Meshing Primitives	100
Triangle Primitive.....	100
Adaptive Surface Meshing	101
Boundary Layer Tool	105
Meshing the Surface	107
▼ Volume Meshing	107
Scheme Designation	108
Mapping	108
Sweeping (Project, Translate, and Rotate)	110
Project.....	111
Translate	112
Rotate	113
Plastering	113
Whisker Weaving	113
Meshing the Volume.....	114
▼ Mesh Duplication	114
▼ Mesh Editing	115
Smoothing.....	115
Surface	115
Volume	116
Accessing Smooth Functions in the GUI	116
Node Repositioning	117

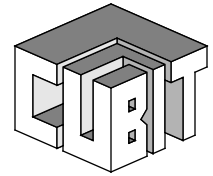
Mesh Deletion	117
Face Deletion	118
Import Mesh	119
▼ Mesh Quality	119
Background	119
Command Syntax	120
Command Examples	121
Example Output	121
Limitations and Planned Enhancements:	121
 Chapter 6: Finite Element Model Definition and Output	123
▼ Finite Element Model Definition	123
Element Blocks	123
Nodesets	124
Sidesets	124
▼ Element Block Specification	124
Default Element Types, Block IDs, and Attributes	125
Element Block Definition Examples	125
Multiple Element Blocks 125	
Surface Mesh Only 125	
Two-Dimensional Mesh 125	
▼ Boundary Conditions—Nodesets and Sidesets	126
Nodeset Associativity Data	126
▼ Setting the Title	127
▼ Exporting the Finite Element Model	127
 Appendix A: Command Index	129
▼ Command Syntax	129
▼ Commands	129
 Appendix B: Examples	147
▼ General Comments	147
▼ Simple Internal Geometry Generation	148
▼ Octant of Sphere	149
▼ Airfoil	151
▼ The Box Beam	152
▼ Thunderbird 3D Shell	155
▼ Assembly Components	158
 Appendix C: Fsqacs: A FASTQ to ACIS Command Interpreter	163

Table of Contents

▼ Description	163
▼ Program Execution	163
▼ Limitations	164
 Appendix D: CUBIT Installation	 165
▼ Licensing	165
▼ Distribution Contents	166
▼ Installation	166
▼ HyperHelp Installation	166
System Requirements	167
CPU	167
Disk Space	167
Printer	167
Operating System	167
Windowing Environment	168
Copying HyperHelp Files	168
Setting Up the HyperHelp Environment	168
 Appendix E: Available Colors	 171
 Appendix F: CUBIT Application Defaults File	 175
 Appendix G: HyperHelp Viewer	 177
▼ Starting the Viewer	177
▼ Using Menus and Buttons	177
Using HyperHelp Menus	177
Using HyperHelp Buttons	178
Using the Keyboard with HyperHelp	178
Menus 178	
Buttons 179	
Scrolling Help Window 179	
▼ Searching for Specific Information	179
To Access Help Topics from the Contents List	179
To Search for a Keyword	180
To Find Any Text	180
▼ Navigating Through Help	181
To Display a Pop-up Definition	181
To Jump to a New Topic	181
To Browse Through a Series of Topics	182
To Define a Bookmark	182
To Go To a Bookmark Topic	182

To Delete a Bookmark	182
▼ Making Notes on Topics	183
To Create an Annotation.....	183
To View an Annotation.....	183
To Delete an Annotation.....	183
▼ Printing Help Topics	184
To Print the Current Help Topic.....	184
To Print All Help Topics.....	184
To Print Selected Help Topics	184
To Configure a Printer	184
 References	 187
 Glossary	 189
 Index	 193
 Appendix H: ERRATA — May 26,1994	 211
Chapter 1.....	211
Chapter 2.....	211
Chapter 3.....	211
Chapter 5.....	211
Appendix A (Command Index)	211
Appendix F.....	212

Table of Contents

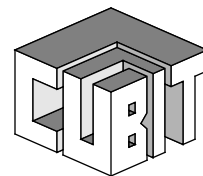


▼ List of Figures

Figure 2-1	Geometry for Cube with Cylindrical Hole.....	28
Figure 2-2	Generated Mesh for Cube with Cylindrical Hole	28
Figure 3-1	Main GUI Window Showing File Menu	41
Figure 3-2	Journal Record/Play Dialog Box	42
Figure 3-3	File Selection Dialog Box.....	43
Figure 3-4	Graphics Mode Dialog Box	45
Figure 3-5	Schematic of From, At, Up, and Perspective Angle	47
Figure 3-6	The Graphics View Dialog Box.....	48
Figure 3-7	Graphics Draw Dialog Box.....	51
Figure 3-8	Visibility Dialog Box.....	52
Figure 3-9	Color Dialog Box.....	54
Figure 3-10	Hardcopy Output Dialog Box	55
Figure 4-1	Cellular Topology Between Volumes.....	69
Figure 4-2	Dangling Faces & Edges.....	69
Figure 4-3	CUBIT Geometry Primitives	70
Figure 4-4	Brick Creation Dialog Box	71
Figure 4-5	Cylinder Creation Dialog Box	72
Figure 4-6	Prism Creation Dialog Box.....	72
Figure 4-7	Frustum Creation Dialog Box.....	73
Figure 4-8	Pyramid Creation Dialog Box.....	74
Figure 4-9	Sphere Creation Dialog Box	75
Figure 4-10	Torus Creation Dialog Box.....	75
Figure 4-11	Sketch Creation Dialog Box	76
Figure 4-12	Body Copy Dialog Box.....	79
Figure 4-13	Body Move Dialog Box	80
Figure 4-14	Body Scale Dialog Box.....	80
Figure 4-15	Body Rotate Dialog Box.....	81
Figure 4-16	Body Reflect Dialog Box.....	81
Figure 4-17	Intersect Boolean Dialog Box.....	82
Figure 4-18	Subtract Boolean Dialog Box	83
Figure 4-19	Web Cutting Dialog	84
Figure 4-20	Solid Model Prior to Decomposition	86
Figure 4-21	Solid Model After Decomposition.....	86
Figure 4-22	Geometry Consolidation Dialog Box.....	87
Figure 5-1	Local Node Numbering for CUBIT Element Types.....	92
Figure 5-2	Illustration of Angle Types	93
Figure 5-3	Scheme Map Logical Properties	94
Figure 5-4	Scheme Submap Logical Properties	95
Figure 5-5	Curve Meshing With The GUI Mesh Dialog Box	96
Figure 5-6	Equal and biased curve meshing.....	98
Figure 5-7	Surface Meshing with the GUI Mesh Dialog	99

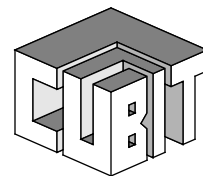
List of Figures

Figure 5-8	Mapped and paved surface meshing	100
Figure 5-9	Submapping Example	101
Figure 5-10	Alternate Submapping Topology Interpretation	101
Figure 5-11	Triangle primitive mesh	102
Figure 5-12	Curvature sizing function meshes on cylinders with varying radii.....	102
Figure 5-13	Illustration of No Sizing, Linear, Interval, and Inverse Sizing Functions	102
Figure 5-14	Test sizing function mesh.	103
Figure 5-15	Plastic strain metric.....	104
Figure 5-16	Adaptively generated mesh.....	105
Figure 5-17	Boundary Layer Parameters.....	105
Figure 5-18	Boundary Layer Dialog Box	106
Figure 5-19	Volume Meshing with the GUI Mesh Dialog.....	108
Figure 5-20	Volume Mapping of an 8-Surfaced Volume.....	109
Figure 5-21	Volume Mapping of a 5-Surfaced Volume.....	109
Figure 5-22	Surface Mesh of an 8-Surfaced Volume Highlighting the Logical Edges Used For Volume Mapping.	110
Figure 5-23	Project Volume Meshing	111
Figure 5-24	Multiple Surface Project Volume Meshing	112
Figure 5-25	Plastering Examples.....	113
Figure 5-26	Whisker Weaving meshes.....	114
Figure 5-27	Smooth Surface and Smooth Volume Dialog Boxes.....	117
Figure 5-28	Mesh Delete and Mesh Delete Warning Dialog Boxes	118
Figure 5-29	Illustration of Quadrilateral Shape Parameters (Quality Metrics)	120
Figure 5-30	Illustration of Quality Metric Graphical Output	122
Figure 6-1	Printer Setup Dialog.....	185
Figure 6-2	Printer Options Dialog	186



▼ List of Tables

Table 3-1	Command Line Interface Line Editing Keys	40
Table 3-1	CUBIT Journal file used for List Output Examples	56
Table 3-2	Sample Output from 'List Model' Command	57
Table 3-3	Sample Output from 'List Names' Command	58
Table 3-4	Sample Output from 'List Group' Command	58
Table 3-5	Sample Output from 'List Body' Command	58
Table 3-6	Sample Output from 'List Volume' Command	59
Table 3-7	Sample Output from 'List Surface' Command	59
Table 3-8	Sample Output from 'List Curve' Command	60
Table 3-9	Sample Output from 'List Vertex' Command	60
Table 3-10	Sample Output from 'List Hex' Command	60
Table 3-11	Sample Output from 'List Face' Command	61
Table 3-12	Sample Output from 'List Edge' Command	61
Table 3-13	Sample Output from 'List Node' Command	61
Table 3-14	Sample Output from 'List Block' Command	61
Table 3-15	Sample Output from 'List SideSet' Command	62
Table 3-16	Sample Output from 'List NodeSet' Command	62
Table 3-17	Sample Output from 'List Settings' Command	63
Table 3-18	Sample Output from 'List View' Command	64
Table 3-19	Sample Output from 'List Memory' Command	64
Table 5-1	Default Meshing Attributes	90
Table 5-2	Valid Meshing Schemes for Curves, Surfaces, and Volumes	91
Table 5-1	Listing of logical sides	94
Table 5-1	Sample Output for 'Quality' Command	121
Table 5-1	Element Quality Plot Legend	121
Table B-1	CUBIT Features Exercised by Examples.	148
Table D-1	HyperHelp Distribution Files	167
Table 6-1	Available Colors	171



Chapter 1: Getting Started

- ▼ How to Use This Manual...17
- ▼ CUBIT Mailing List...18
- ▼ Problem Reports and Enhancement Requests...18
 - ▼ Executing CUBIT...19
 - ▼ Command Syntax...22
 - ▼ Features...24
 - ▼ Future Releases...26

Welcome to CUBIT, the Sandia National Laboratory automated mesh generation environment. With CUBIT the geometry of a part can be imported, created, and/or modified using an embedded solid modelling engine. The geometry can then be discretized into a finite element mesh using a combination of techniques including paving [1], mapping, sweeping, and various other algorithms being developed. CUBIT also features boundary layer meshing specifically designed for fluid flow problems. Boundary conditions can be applied to the mesh through the geometry and appropriate files for analysis generated. CUBIT is specifically designed to reduce the time required to create all-quadrilateral and all-hexahedral meshes.

▼ How to Use This Manual

This manual provides specific information about the commands and features of CUBIT. It is divided into chapters which roughly follow the process in which a finite element model is designed, from geometry creation to mesh generation to boundary condition application. An example is provided in a tutorial chapter to illustrate some of the capabilities and uses of CUBIT. Appendices containing complete command usages, examples, installation instructions, and a list of available colors are included.

The CUBIT environment is designed to provide the user with powerful meshing algorithms that require minimal input to produce a complete finite element model. As such, the code is constantly being updated and improved. Feedback from our users indicates that new meshing tools are often needed and/or desired before they have been completely tested and debugged. As a service to the user, these tools are integrated and made available as quickly as possible, but in a *user beware* state. As they are further tested (often with the assistance of users) and improved,



the state of the particular tool becomes more stable, and thus the risk to the user is lowered. Since documentation of the tool is necessary for actual use, we have included the documentation of all available tools in the manual. However, to warn the user, a “hammer” icon is placed in the document next to those features that are only minimally tested or are in a state of work-in-progress (See “hammer” icon in left margin). In other words, “proceed with caution.” Certain portions of this manual contain information that is vital to understand in order to run CUBIT effectively. In order to highlight these portions, a “key” icon is positioned in the document next to these sections. In other words, “this is a key point”.

This manual is Volume 1 of the CUBIT documentation set. The companion document is *CUBIT Mesh Generation Environment, Volume 2: Developers Manual* [3] which contains internal programming-related details of the CUBIT mesh generation environment.

This manual documents CUBIT Version 1.11.0, March 20, 1995.

▼ CUBIT Mailing List

A mailing list has been created to keep interested users informed of new features, bug-fixes, and other pertinent information about CUBIT. The list can also be used by users for general discussions about CUBIT. Users can subscribe to the mailing list by sending a mail message to **listserv@sahp046.jal.sandia.gov** with the body (not the subject) of the mail message containing the line:

subscribe cubit Your Full Name

The user would then receive a message confirming the subscription to the CUBIT mailing list. More information about the use of the mailing lists can be obtained by sending the message **help** to the above mail address. Messages are sent to the list by sending mail to the address:

cubit@sandia.gov

The CUBIT developers will be sending announcements of new CUBIT capabilities, enhancements, and user-visible bug fixes to this list on a regular basis. In addition, this list may be used for general questions regarding CUBIT that may be solvable by other subscribers to the list.

An additional mailing list has been created for direct communication with the CUBIT developers. All messages sent to this list will be distributed to the CUBIT developers only. It should be used for questions that are not of general interest to other CUBIT users. Messages are sent to the CUBIT developers by sending mail to the address:

cubit-dev@sandia.gov

▼ Problem Reports and Enhancement Requests

The CUBIT project is using GNATS [17], the GNU Problem Report Management System, for tracking questions, problem reports, and enhancement requests. The GNATS system is designed to allow users who have problems with the CUBIT code, or who have requests for new features to be added to the CUBIT code, to submit reports of these problems or requests to the CUBIT developers. The GNATS system provides a program called *send-pr* which can be used to submit

problem reports and enhancement requests in a standard, defined format which can be read by an electronically managed database which automatically notifies responsible parties of the existence of the problem. It also provides a mechanism for keeping everyone involved in the problem report informed of the current state of the problem.

In general, any editor and mailer can be used to submit valid Problem Reports (PR), as long as the format required by GNATS is preserved. However, *send-pr* automates the process and ensures that certain fields necessary for automatic processing are present. The use of *send-pr* is strongly recommended for all initial problem-oriented correspondence with the CUBIT developers. The user documentation for the GNATS system is supplied with the CUBIT program. On the Sandia Internal Restricted Network, the documentation is available from `sahp046.jal.sandia.gov:/usr/local/doc/gnats.ps`. On the Sandia External Open Network, the documentation is available from `sass577.endo.sandia.gov:/home/cubit/gnats.ps`. Contact your CUBIT code sponsor, or if this fails, Greg Sjaardema (`gdsjaar@sandia.gov`) if you cannot access either of these locations or if you have problems using the system.



Note: The existence and recommended use of an electronic bug reporting mechanism is not an attempt by the CUBIT developers to ignore and or discourage face-to-face discussion of problems with, or enhancements to the CUBIT code with users. The use of GNATS is intended to help the developers manage, prioritize, and track the tasks required to produce a usable “state-of-the-art” production mesh generation package.

▼ Executing CUBIT

Execution Command Syntax

Two versions of CUBIT are currently supported: 1) a basic command line version which output graphics to a standard X Window System graphics window, and 2) a batch command line version with no graphics. The commands to execute these versions of CUBIT on most systems are simply:

cubit Command line version with X Window system graphics.

cubitb Batch command line version, no graphics¹.

Throughout this manual, “CUBIT” will be used as a generic term that applies to all of the executables. If it is necessary to specify a specific version of CUBIT, one of above names will be used.

The command syntax recognized by CUBIT is:

```
{cubit|cubitb} [-help] [-initfile <val>] [-noinitfile] [-solidmodel <val>]
[-batch] [-nojournal] [-journalfile <file>] [-maxjournal <val>]
[-noecho] [-debug=<val>] [-information={on|off}] [-warning={on|off}]
[-Include <path>] [-fastq <fastq_file>] {<input_file_list>|<var=value>}...
```

1. The `cubitb` executable may be eliminated in the future and its functionality duplicated using the ‘`-batch -nographics`’ command line options to the `cubit` executable.

where the quantities in square brackets **[-options]** are optional parameters that are used to modify the default behavior of CUBIT and the quantities in angle brackets **<values>** are values supplied to the option. The effect of these parameters are:

-help Print a short usage summary of the command syntax to the terminal and exit.

-initfile <val> Use the file specified by **<val>** as the initialization file instead of the default initialization file **\$HOME/.cubit**.

-noinitfile Do not read any initialization file. The default behavior is to read the initialization file **\$HOME/.cubit** or the file specified by the **-initfile** option if it exists.

-solidmodel <val> Read the ACIS solid model geometry information from the file specified by **<val>** prior to prompting for interactive input.

-batch Specify that there will be no interactive input in this execution of CUBIT. CUBIT will terminate after reading the initialization file, the geometry file, and the file specified by the **-initfile** option.

-nojournal Do not create a journal file for this execution of CUBIT. This option performs the same function as the **Journal Off** command. The default behavior is to create a journal file.

-journalfile <file> Write the journal entries to the file **<file>**. The file will be overwritten if it already exists.

-maxjournal <val> Only create a maximum of **<val>** default journal files. Default journal files are of the form **cubit.#.jou** where # is a number in the range 01 to 99.

-noecho Do not echo commands to the console. This option performs the same function as the **Echo Off** command. The default behavior is to echo commands to the console.

-debug=<val> Set the debug message flags indicated by **<val>**. **<val>** is a comma-separated list of integers or ranges of integers. An integer range is specified by separating the beginning and the end of the range by a hyphen. For example, to set debug flags 1, 3, and 8 to 10 on, the syntax would be **-debug=1,3,8-10**. Flags not specified are off by default. Debug messages are typically of importance only to developers and are not normally used in normal execution.

-information={on|off} Turn on/off the printing of information messages from CUBIT to the console.

-warning={on|off} Turn on/off the printing of warning messages from CUBIT to the console.

-Include=<include_path> Set the patch to search for journal files and other input files to be **<include_path>**. This is useful if you are executing a journal file from another directory and that journal file includes other files that exist in that directory also.

-fastq=<fastq_file> Read the mesh and geometry definition data in the FASTQ file **<fastq_file>** and interpret the data as FASTQ commands. See Reference [5] for a description of the FASTQ file format.

The information following the last option on the command line consists of either input files or variable definitions. Variables are specified by the syntax **<variable=value>** where variable is any valid variable name (See Reference [13]) and value is either a real value or a string value. String values must be surrounded by double quotes. Input files are specified simply by typing the filename. All files specified on the command line following the last option are processed in the order they are listed prior to prompting for interactive command input.

An example of the use of the command line options is:

```
cubitb -batch -nojournal final_mesh.jou height=1.2345
```

which specifies that **cubitb** will execute the commands in the file **final_mesh.jou** unattended. The variable **height** will be defined to have the value 1.2345. This mode is typically used to recreate a previously generated mesh with no user interaction.

The command options can also be specified through the **CUBIT_OPT** environment variable. See the “User Environment Settings” section below for more information.

Initialization File

If the file **\$HOME/.cubit** or the file specified by the optional **-initfile <val>** option exists when CUBIT begins executing, it is read prior to beginning interactive command input. This file is typically used to perform initialization commands that do not change from one execution to the next, such as turning off journal file output, setting geometric and mesh entity colors, and setting the size of the graphics window.

User Environment Settings

To execute CUBIT several environment variables must be set. In particular the “**HOME**”, “**PATH**” “**HOOPS_PICTURE**” and “**DISPLAY**” variables. The **HOME** environment variable is typically set automatically when you login to a system. Its purpose is to provide a pointer to your login directory. The **PATH**, on a Unix system, is a list of directories that are searched for commands to be executed. Proper setting of the path is system-dependent; if CUBIT does not execute correctly, contact your system manager or another CUBIT user for the correct setting of the **PATH** specification.

The X Window System-based command line input version of CUBIT (**cubit**) requires the specification of the **DISPLAY** and **HOOPS_PICTURE**¹ environment variables which are used by the application to determine where the graphics window should be displayed (and which screen should be used on displays with multiple monitors). C Shell users can set the environment variable by typing:

```
setenv HOOPS_PICTURE          x11/my_display:0.1
```

This will make the Graphics Window show up on display number 0.1.

CUBIT also requires the environment variable **CUBIT_HELP_DIR** if the online hypertext help system will be used. This variable should be set to the pathname specifying where the CUBIT help file **cubitHelp.hlp** is located. The person responsible for installing CUBIT on the system should be contacted for this information.

Another useful environment variable is **CUBIT_OPT** which can be used to set execution command line parameter options. For example, if journalling of commands is never wanted, then setting **CUBIT_OPT** to **-nojournal** will turn off journalling for all CUBIT executions².

Graphics Customization

Settings for the default CUBIT window sizes, locations, colors, and fonts can be set in the **.Xdefaults** or **.Xresources** resource files in the user’s home directory. This file is a text file that can be edited with any standard UNIX text editor. In the resource file, each resource must

1. **HOOPS_PICTURE** will automatically be set to **x11/\$DISPLAY** if it is not set by the user.
2. Journalling could then be turned back on with the “Record” command.

be on a separate line. The resource setting consists of a resource label, a colon, one or more spaces or tabs, and the resource value. For additional general information on resources, see X Window System documentation; a readily available documentation source is the RESOURCES section of the X(1) manual page which is usually installed on most systems.

A CUBIT resource label begins with the word “cubit” followed by an asterisk or period, followed by the specification of the resource. For example, to specify that the CUBIT graphics window should be 700 pixels square rather than the default size, the following line should be added to the resource file:

```
cubit*CUBIT.geometry: 700x700+445+0
```

Another file, similar to the resource file is the Application Defaults file which is used to customize CUBIT’s graphical user interface. This file, called **CUBIT.ad** is distributed with the CUBIT executables. This file must be renamed to **CUBIT** and then installed in either `/usr/lib/X11/app-defaults` for system-wide defaults and/or in a user’s home directory for per-user defaults. The contents of this file are reproduced in “CUBIT Application Defaults File” on page 175. The format of this file is the same as the resource file format.

For example:

```
cubit*XmTextField*background: LightBlue
```

This line states that the background of all text fields in the **cubit** application will be the color LightBlue. Colors can normally be found in the `/usr/lib/X11/rgb.txt` file on your system. For additional information about application default files, see the application defaults section in any X Window user’s book.

▼ Command Syntax

The execution of CUBIT is controlled either by inputting commands from the command line (or in the command line field in the main window of the Motif-based version of CUBIT [See Figure 3-1 on page 57]), or through the use of the Graphical User Interface (GUI). The Graphical User Interface parses the user input and generates an equivalent command which is recorded in a journal file and actually executes the action requested. Throughout this document, each function or process will have a description of both the command required to perform the function or process and the steps required to perform the same function through the graphical user interface. In this section, the command syntax used in this manual will be described. Although knowledge of the command syntax is not necessary for the use of the graphical user interface version of CUBIT, it may be helpful to skim this section anyway since the journal file written by CUBIT uses these commands. The user can obtain a quick guide to proper command format by issuing the **<keyword> help** command. This help message will indicate the full command syntax expected by the keywords. For example, entering **view help** results in the following output:

```
View At <X_coord> <Y_coord> <Z_coord>
```

```
View From <X_coord> <Y_coord> <Z_coord>
```

```
View List
```

```
View Up <X_coord> <Y_coord> <Z_coord>
```

The words that begin with an uppercase letter are keywords which must be entered (case is not significant) and the bracketed words are user supplied parameters.

The commands recognized by CUBIT are free-format and must adhere to the following syntax rules.

- Either lowercase or uppercase letters are acceptable.
- The “#” character in any command line begins a comment. The “#” and any characters following it on the same line are ignored.

Each command typically has either:

- an action keyword or “verb” followed by a variable number of parameters, for example

Mesh Volume 1

- or a selector keyword or “noun” followed by a “verb” or “selector keyword and a variable number of parameters, for example

Volume 1 Scheme Project Source 1 Target 2

The action or selector keyword is a character string matching one of the valid commands. It may be abbreviated as long as enough characters are used to distinguish it from other commands. The meaning and type of the parameters depend on the keyword. Valid entries for parameters are:

- A numeric parameter may be a real number or an integer. A real number may be in any legal C or FORTRAN numeric format (for example, 1, 0.2, -1e-2). An integer parameter may be in any legal decimal integer format (for example, 1, 100, 1000, but not 1.5, 1.0, 0x1F).
- A string parameter is a literal character string contained within single or double quotes. For example, **‘This is a string’**.
- A filename parameter must specify a legal filename on the system that CUBIT is running. Environment variables and aliases may not be used in the filename specification. For example, the C-Shell shorthand of referring to a file relative to the user’s login directory (**~jdoe/cubit/mesh.jou**) is not valid. The filename must be specified using either a relative path (**../cubit/mesh.jou**), or a fully-qualified path (**/home/jdoe/cubit/mesh.jou**). Like a string, it also must be contained within single quotes.
- Several commands permit a range of values. A range is one of the following forms:
 - “**n1**” selects a single value **n1**,
 - “**n1 to n2**” selects all values from **n1** to **n2**. The value **n2** must be greater than **n1**,
 - “**n1 to n2 by n3**” selects all values from **n1** to **n2** stepping by **n3**, where **n3** may be positive or negative.

The keywords “**through**” and “**thru**” may be used instead of “**to**,” and “**step**” may be used instead of “**by**.”
- Some commands require a “toggle” keyword to enable or disable a setting or option. Valid toggle keywords are “**on**”, “**yes**”, and “**true**” to enable the option; and “**off**”, “**no**”, and “**false**” to disable the option.

The notation conventions used in the command descriptions in this document are:

- The command will be shown in a format that **looks like this**,
- A word enclosed in angle brackets (**<parameter>**) signifies a user-specified value. The value can be an integer, a range of integers, a real number, or a string. The valid value types should be evident from the command or the command description.
- A series of words in braces and delimited by a vertical bar (**{choice1 | choice2 | choice3}**) signifies that one of the words within the brackets must be entered.
- A word enclosed in square brackets (**[optional]**) signifies an optional parameter which can be entered to modify the default behavior of the command, but is not required.

An example of this command syntax is shown below.

{volume surface curve} <range> size <size>	
volume 1 size 0.5	Valid
surface 1 to 10 by 3 size 0.05	Valid
volume 10 to 1 size 0.05	Invalid — negative increment
volume 10 to 1 by -1 size 0.05	Valid
surface 1 10 size 1.0	Invalid — not a valid range specification
surface 1 to 10 interval 5.0	Invalid — “interval” requires an integer

▼ Features

The CUBIT environment is designed to provide the user with powerful meshing algorithms that require minimal input to produce a complete finite element model. CUBIT is based on a solid modeler that provides it with a precise geometric representation. The *paving* algorithm [1] has been extended to mesh complex three dimensional surfaces based on the solid modeler. Volumetric meshing is provided by mapping transformations and sweeping algorithms. Multiple user interfaces are supported as are several quadrilateral and hexahedral element types. The following sections provide a brief overview of the CUBIT meshing toolkit.

Geometry Creation

Geometry creation is accomplished using the geometric primitives and boolean operations in CUBIT or by reading an external solid model file into the CUBIT meshing toolkit. External solid model files can be created from any of several environments that support the ACIS® solid model format: a rudimentary command line system, referred to as the “test harness [4],” is useful for building quick and straightforward models. Other more advanced environments include the Aries® ConceptStation and PRO/Engineer via a PRO/Engineer/ACIS translator. A specialized software translator has been designed to translate sheet solid models from FASTQ [5] input files into an ACIS format, which can be further modified using the ACIS test harness. The resulting ACIS models can then be imported into CUBIT and meshed.

Algebraic Command Preprocessing

Many analysts use the Aprepro [13] program to preprocess commands and journal files which contain algebraic expressions. The Aprepro algebraic preprocessing capability has been implemented into the CUBIT command parser. The full Aprepro functionality has been included except for the **units**, and **loop** commands.

Geometry Consolidation

When assembly solid models are imported into the CUBIT environment, many surface, curve, and vertex entities will be redundant. To resolve this issue, the automated geometry consolidation or “merge” routines will identify matching entities and make database modifications to remove the redundancy. Geometry consolidation can also be interactive, in case certain redundant features need to be retained to represent slide surfaces or slide lines. The

geometry merge capability eliminates the generation of non-contiguous elements between adjacent surfaces and curves which would have to be removed after meshing.

Geometry Decomposition

Solid models imported into the CUBIT environment sometimes consist of combinations of simple geometric volumes, for example a plate with a cylinder projecting out of it. These geometries are also sometimes constructed within CUBIT. Currently these geometries must be decomposed into topologically primitive lumps (cylinder, brick, etc) before being able to be mesh. CUBIT contains functions which aid in the decomposition of complicated geometries into meshable pieces.

Supported Element Types

Element types supported in CUBIT include 2 and 3 node bars and beams; 4, 8, and 9 node quads; 4, 8, and 9 node shells; and 8, 20, and 27 node hex elements. Element types must be set before mesh generation is initiated.

Mesh Creation

Mesh generation in CUBIT is designed to be highly automated although numerous control mechanisms are provided to allow the user to guide the meshing process. Meshing is controlled through scheme choice, and interval number or node density specification. Curve meshing schemes include equally spaced and biased spaced intervals. Surface meshing schemes include mapping transformations, paving, boundary layers, and primitives. Volume meshing schemes include mapping transformations and mesh sweeping or projecting. Other automated volume meshing algorithms are being added.

Boundary Condition Application

Once a suitable mesh has been generated, elements can be grouped into sets using three control classes: element blocks, nodesets, and sidesets. Numeric flags are associated with these sets allowing analysis codes to apply appropriate boundary conditions to the correct mesh entities.

Element blocks are used for efficient storage of a finite element model. Within an element block, all elements are of the same type (basic geometry and number of nodes) and have the same material definition.

Nodesets provide a means to reference a group of nodes with a single identification number rather than by each node's identification number. Nodesets are typically used to specify load or boundary conditions, or to identify a set of nodes for a special processing within CUBIT. A node may appear in multiple nodesets, but will only appear once in any single nodeset.

Sidesets provide an additional means of applying load and boundary conditions. Unlike nodesets, sidesets group sides or faces of elements rather than simply a list of nodes. For example, a pressure load must be associated with elements rather than nodes to apply it properly.

Nodes, element edges, and element faces can belong to multiple nodesets and sidesets. Nodesets and sidesets can be individually displayed for visual inspection. See reference [6] for more information.

Graphical Display Capabilities

CUBIT uses the Hoops graphic display environment to render images. CUBIT can display a wireframe, hiddenline, or shaded representation of geometric and mesh entities. CUBIT can also generate a PostScript file of any displayed image (see “Hardcopy Output” on page 55). Complete control over the viewport parameters and the zoom magnification provide the user with an intuitive modeling environment.

When operating the GUI, users can perform screen picking and point-and-click zoom operations. All of the user-defined options are represented inside option windows by colored status buttons. This gives the user an easy to read description of the current settings. The GUI also gives the user control over the display of the geometry and mesh.

Hardware Platforms

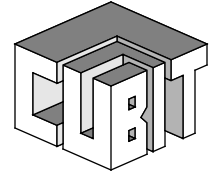
CUBIT is written in “standard” C++ and should execute on any Unix operating system. To date, it has been compiled and used on Sun (both SunOS and Solaris), Hewlett-Packard (HP-UX 9.X), and Silicon Graphics workstations (IRIX 5.3).

▼ Future Releases

CUBIT is currently on a 4-month major release cycle (April, August, and December). The capabilities of CUBIT will be expanded and enhanced on a regular basis as dictated by user needs and the developmental progress of new meshing algorithms. Areas of concentration will include

- full-featured automatic hexahedral meshing using a combination of *plastering* and *whisker-weaving*,
- quadrilateral and hexahedral adaptivity,
- enhanced geometric functionality such as overlapping geometry consolidation and more robust geometry decomposition.
- improved control of the naming of geometric and mesh entities through the use of persistent identification numbers or attributes.
- improved internal geometry generation through the use of an interactive sketch pad.
- improved usability, robustness, and functionality.

Extension of the CUBIT environment to new platforms will also be pursued according to user needs.



Chapter 2: Tutorial

▼ The Tutorial...	27
▼ Step 1: Beginning Execution...	29
▼ Step 2: Creating the Brick...	29
▼ Step 3: Create the Cylinder...	31
▼ Step 4: Adjusting the Graphics Display...	32
▼ Step 5: Forming the Hole...	32
▼ Step 6: Setting Body Interval Size...	33
▼ Step 7: Setting Specific Surface Intervals...	33
▼ Step 8: Setting Specific Curve Intervals...	34
▼ Step 9: Surface Meshing...	35
▼ Step 10: Volume Meshing...	35
▼ Congratulations!...	37

The purpose of this chapter is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a brief tutorial on the use of the software package. This chapter is designed to demonstrate step-by-step instructions on generating a simple mesh on a perforated block.

▼ The Tutorial

The following is a sample of the basics of using CUBIT to generate and mesh a geometry. By following this tutorial, you will become familiar with the command-line interface and with as much of the CUBIT environment as possible without stopping for detailed explanations. All the commands introduced in this tutorial are thoroughly documented in subsequent chapters. Here are a few tips in following the example in the tutorial

- Focus on instructions preceded with “Step” numbers. These step you through a series of explicit activities that describe exactly what to do to complete the task.
- Refer to screen shots and other pictures that show you what you should see on your own display as you progress through the tutorial.

- An example of the command line is shown below. In this tutorial, the command that you should type will be preceded by the word “Command” and a colon.

Command: This is a Command Line

The example given in this tutorial will demonstrate the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. The Create, Brick, Cylinder and Subtract commands are used to create solid model geometry with primitives and boolean operations. The block is then meshed using paving and translation. The geometry to be generated is shown in Figure 2-1. This figure also shows the curve and surface identification (ID) numbers of the

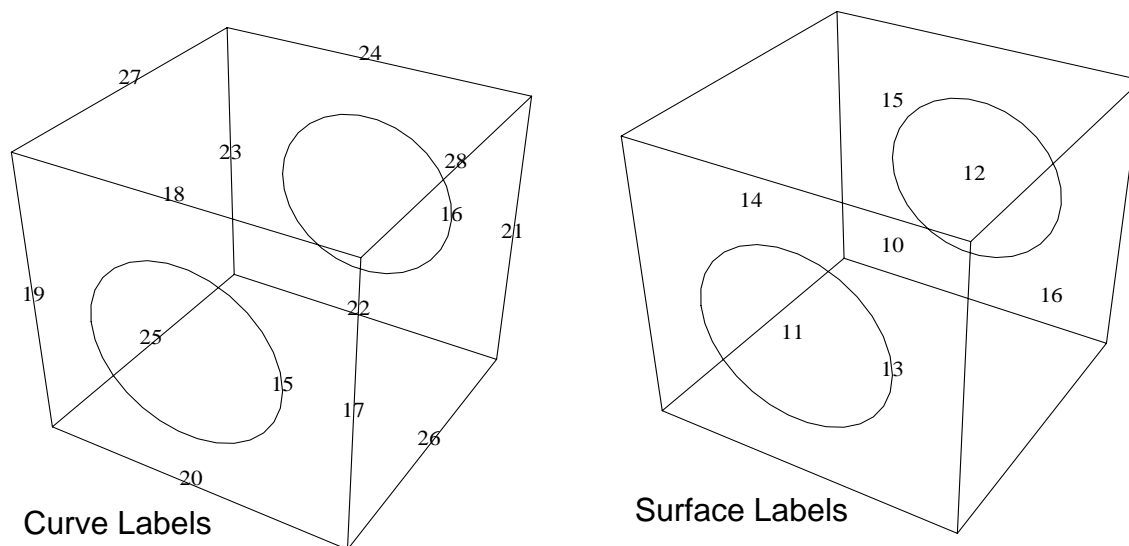


Figure 2-1 Geometry for Cube with Cylindrical Hole

geometry. These ID numbers are used in the command lines shown with each step. The final meshed body is shown in Figure 2-2 and also at the end of this chapter.

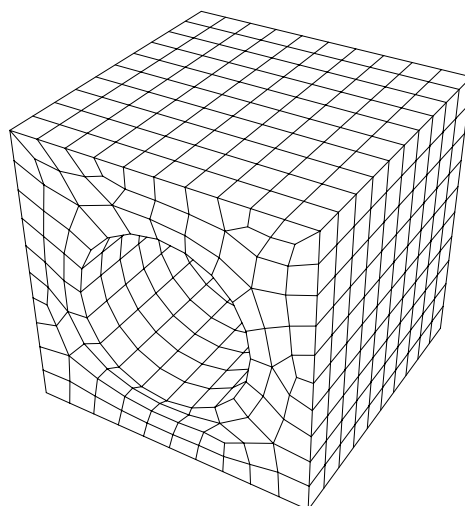


Figure 2-2 Generated Mesh for Cube with Cylindrical Hole

▼ Step 1: Beginning Execution

- Type “cubit” to begin execution of CUBIT. If you have not yet installed CUBIT, see instructions for doing so in the “CUBIT Installation” Appendix. A CUBIT console window will appear which tells the user which version is being run and the most recent revision date. (See the following screen shot for example of window). This window relays information about the success or failure of attempted actions.

```

      CCCCC      UU      UU      BBBBBB      IIII      TTTTTT
      CC      CC      UU      UU      BB      BB      II      TT
      CC      UU      UU      BB      BB      II      TT
      CC      UU      UU      BBBBBB      II      TT
      CC      UU      UU      BB      BB      II      TT
      CC      CC      UU      UU      BB      BB      II      TT
      CCCCC      UUUUU      BBBBBB      IIII      TT

[ *** CUBIT Version 1.8.1 ***

      *** ACIS Version 1.5 ***

      Revised 5/1/94

      AN ALL-QUADRILATERAL AND ALL-HEXAHEDRAL MESH
      GENERATION PROGRAM FOR
      PRE-PROCESSING OF FINITE ELEMENT ANALYSES

      CUBIT is based upon ACIS software by SPATIAL TECHNOLOGY INC.

      Executing on 05/20/94 at 09:37:35

```

- At the bottom of the CUBIT window you will be told where the commands entered in this CUBIT session will be journalled. For example: “Commands will be journalled to ‘cubit01.jou’.
- Below that, you will be offered the command line prompt: “CUBIT>”.
- Commands are entered at that prompt, followed by the “Enter” key.
- You should also see a blank graphics display window.

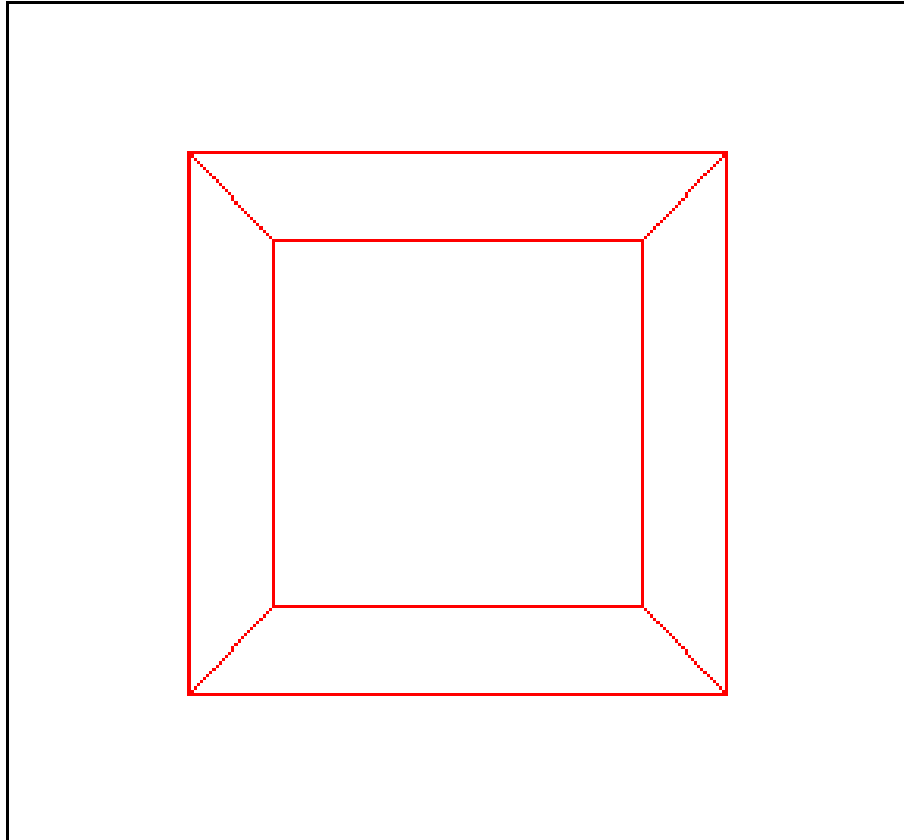
▼ Step 2: Creating the Brick

Now you may begin generating the geometry to be meshed. You will create a brick of width 10, height 10 and depth 10. The width and depth correspond to the x and y dimensions of the object being created. The “width” or x-dimension is screen-horizontal and the “depth” or y-dimension is screen-vertical. The height or z-dimension is into the screen. The command to create an object is **Create**, followed by the type of geometry and its dimensions. Enter the following command.

Command: Create Brick Width 10. Depth 10. Height 10.

- The cube should appear in your display window as shown below.

Brick Display



- Note that the journalled version of the command is echoed above the next command line along with the confirmation message “brick body 1 successfully created.”
- The controller for the command line interface can tell from context when the user wants to create an object, so the command word **Create** can be left out. The same result as above would have been obtained by entering:

Command: Brick Width 10. Depth 10. Height 10.

- Try this after first issuing a **Reset** command as follows to remove all previous geometry and previous mesh from computer memory.

Command: Reset

- The command line interpreter can also recognize shortened versions of commands if they are unambiguous. The following command line would also have worked.

Command: Br Wi 10. Dep 10. Hei 10.

- If the command line parser is not able to find an unambiguous match to one of your command words, it will give an error message and offer a list of possible matches. For clarity, this tutorial will use unabbreviated command words.
- If the brick being created is a true cube, only the width needs to be specified, so the following command would also have worked. Try this after issuing a **Reset** command.

Command: brick width 10

- Notice that the command line is not case-sensitive, so **Brick** and **Width** do not need to be capitalized.

▼ Step 3: Creating the Cylinder

Now you must form the cylinder which will be used to cut the hole from the brick:

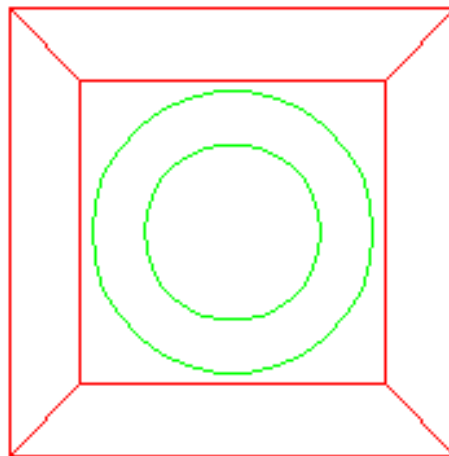
Command: create cylinder height 12 radius 3

or, by leaving the **Create** command as implicitly understood:

Command: cylinder height 12 radius 3

At this point you will see both a cube and a cylinder appear in the CUBIT display window.

Brick with Cylinder Display

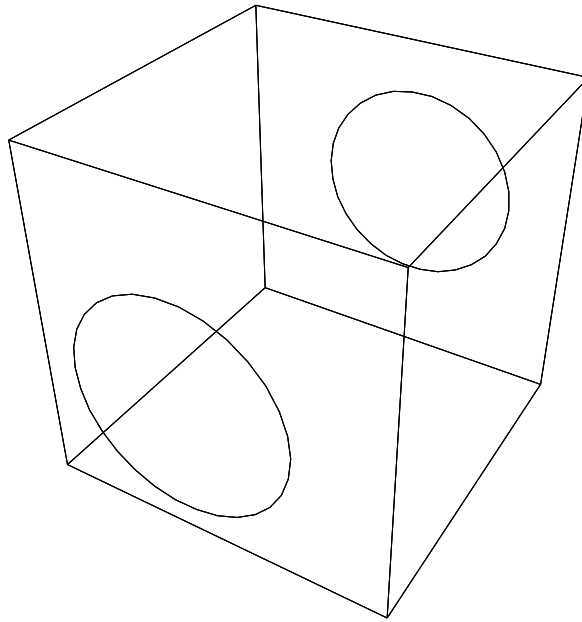


▼ Step 4: Adjusting the Graphics Display

The picture on the graphics display can now be adjusted to verify that what you expected to happen has indeed occurred. Issue the command

Command: from 3 4 5

This changes the viewpoint of the screen to a viewing location along the vector (3,4,5). The command word **From** stands for “view from.” The display should now look like the following figure.



At this point you may rotate the model to view its validity. The easiest method is to use the mouse. Type **Mouse** on the command line. You should see the message: “Entering mouse-based rotation/zooming/panning mode” and a circle should appear around the figure in the graphics window. To rotate the figure about its y-axis, position the mouse pointer outside the circle, hold the left mouse button down and move the mouse pointer around the circle. To rotate it around the x- and z-axes, position the pointer inside the circle and hold the left button down while moving the mouse. To exit the mouse-based mode of rotation, type **q**.

In the display, the wireframe picture shows the relative locations of the bodies. Turning the image to smooth shaded (as will be described in following steps) improves the perspective.

▼ Step 5: Forming the Hole

Now the cylinder can be subtracted from the brick to form the hole in the block. Issue the following commands.

Command: Subtract 2 From 1

Note: Note that both original bodies are deleted in the boolean operation and replaced with a new body (3) which is the result of the boolean operation **Subtract**.

▼ Step 6: Setting Body Interval Size

The next step is to generate the surface mesh on one of the surfaces to be swept. The number of increments must be set before meshing any geometry. This is done first for the entire body by specifying a desired size interval. Recall that body number 3 is a 10 by 10 by 10 cube with a cylindrical hole through it. We decide to specify an interval size of 1, which suggests 10 intervals on each side. Issue the following command.

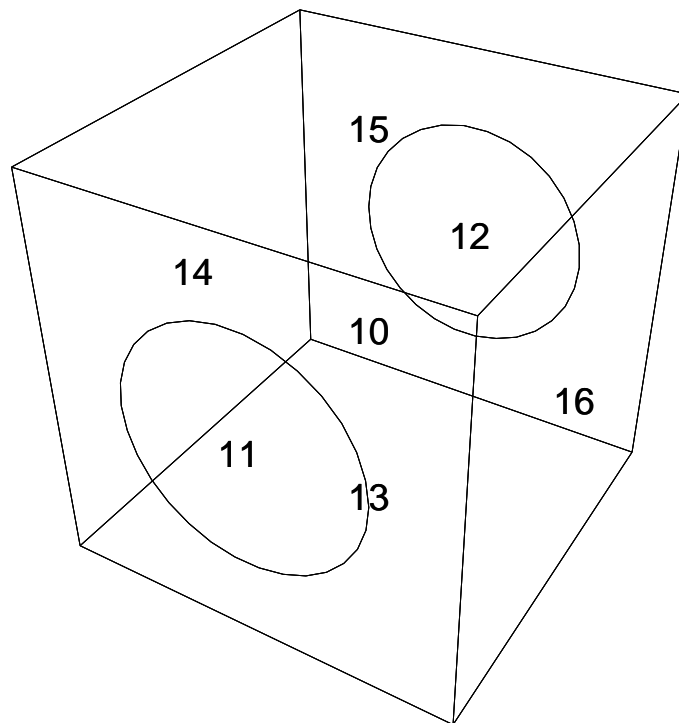
Command: body 3 interval size 1.0

▼ Step 7: Setting Specific Surface Intervals

The cylindrical surface (the inside of the hole) must be mapped in order for the sweeping type tools to work. Since this surface is periodic (contains no edge along the side of the cylinder) the mapping algorithm uses the surface interval setting to determine how many elements are to be mapped along the axis of the cylinder. The surface must first be identified. To see the surface numbers, issue the following commands.

Command: label surface on

Command: display



- The first command turns the surface labels on, but they do not become visible until the **display** command forces an update of the graphics screen. The surface labels can now be seen.

- The surface labels are positioned in the center of the geometric bounding box for each surface. From the display it is evident that the cylindrical surface is surface 10. To set the number of intervals to 10 on this surface only, issue the following command.

Command: surface 10 interval 10

▼ Step 8: Setting Specific Curve Intervals

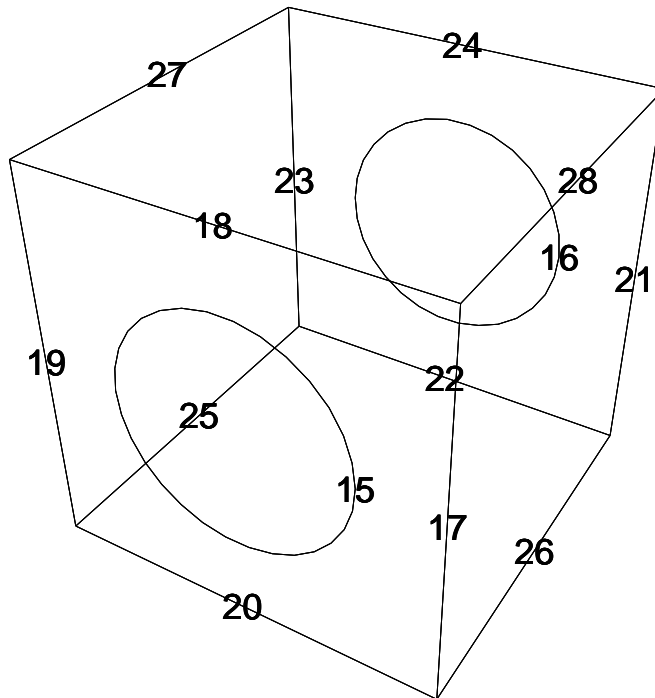
The surface interval command also propagates downward to the edges. However, in this case we want the circumference of the cylindrical ends of the surface to have 20 intervals instead of 10. This is accomplished by issuing a command of the form **Curve m Interval n**, where m is the curve ID and n is 20. Before we can do that, we need to identify the curves. To turn off the surface labels and turn on the curve labels, issue the following commands.

Command: label surface off

Command: label curve on

Command: display

From the graphics display shown below, it is evident that curves 15 and 16 are the correct curve ID's.



To set the number of intervals on these curves to 20, issue the following command.

Command: curve 15 to 16 interval 20

Notice that we specified two curves, 15 and 16, by using the command syntax **15 to 16**.

▼ Step 9: Surface Meshing

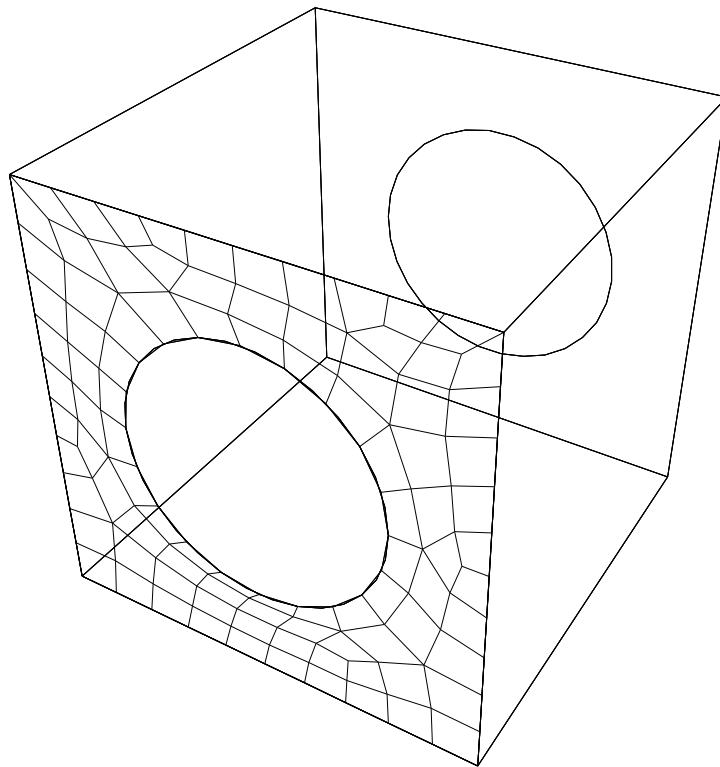
Now all necessary intervals have been set, and the meshing can proceed. Begin by meshing the front surface (with the hole) using the paving algorithm. This is done in two steps. First set the scheme for that surface to **Pave**, then issue the command to **Mesh**. Since the surface to be paved is number 11, issue the command:

Command: surface 11 scheme pave

With the meshing scheme specified, we proceed to mesh the surface.

Command: mesh surface 11

- The result of this meshing operation is shown below.



▼ Step 10: Volume Meshing

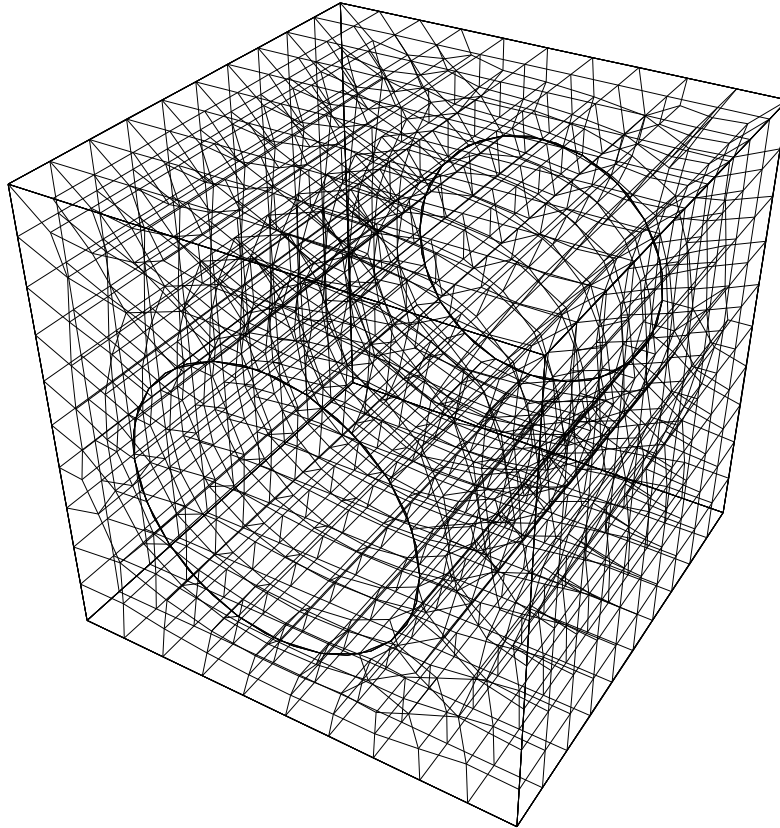
The volume mesh can now be generated. Again, the first step is to specify the type of meshing scheme and the second step is to issue the order to mesh. The scheme chosen is **translate**, which requires that source and target surfaces be specified. Issue the following command.

Command: volume 3 scheme translate source 11 target 12

With the scheme set, the **mesh** command may be given:

Command: mesh volume 3

The final meshed body will appear in the display window



The type, quality, and speed of the rendered image can be controlled in CUBIT by using several **graphics mode** commands, such as **Wireframe**, **Hiddenline**, and **Smoothshade**. For example:

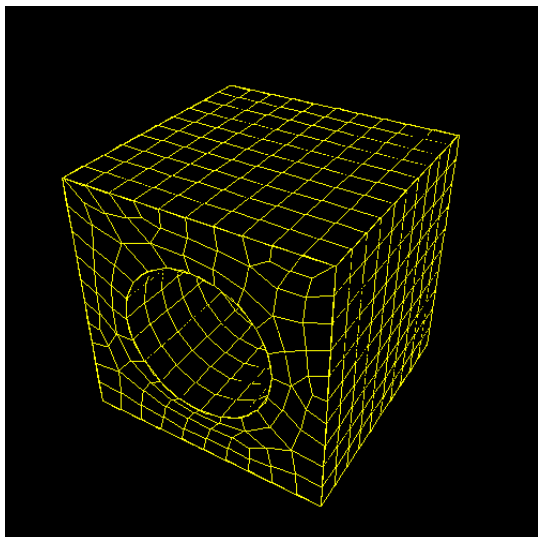
Command: graphics mode hiddenline**Command: display**

The hidden line display is illustrated below. Next, try:

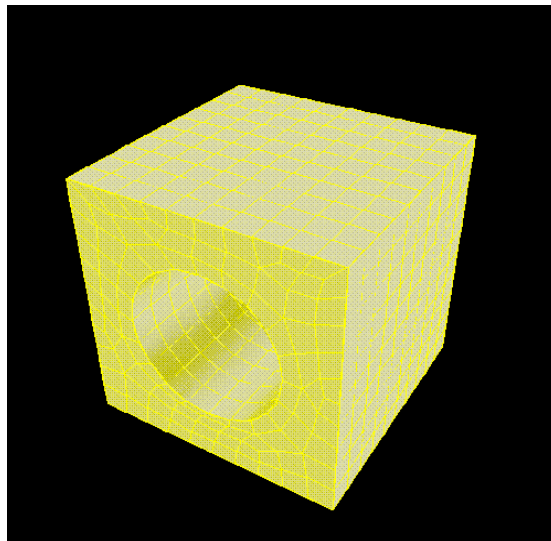
Command: graphics mode smoothshade**Command: display**

The smoothshade display is also shown below.

For detailed information on these, see Chapter 3, Environment, “Image Rendering Control” and “Viewing the Model.”



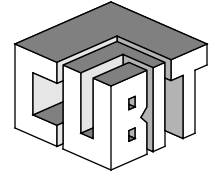
Hiddenline Display



Smoothshade Display

▼ Congratulations!

You have created your first CUBIT mesh. The following chapters contain more detailed information about using CUBIT and an in-depth description of the meshing algorithms available.



Chapter 3: Environment

- ▼ Interface Choices...39
 - ▼ Session Control...41
 - ▼ Journal Files...42
 - ▼ Graphics...44
- ▼ Model Information...56
- ▼ Program Messages...81
- ▼ Help Facility...65

The CUBIT user interface is designed to fulfill multiple meshing needs throughout the analysis process. The user interface options include a GUI based environment, a traditional command line interface, and batch mode operation. This chapter covers the interface options as well as the use of journal files, control of the graphics, a description of ways to interrogate the model for specific information, and an overview of the help facility.

▼ Interface Choices

Overview

The user interface options for CUBIT are: 1) command line, 2) batch mode, and 3) a graphical user interface (GUI). The command line version requires the user to type commands in order to interact with CUBIT. The GUI allows a user to interact with CUBIT by pressing buttons and selecting menu choices, as well as entering commands using a more traditional command line. The batch mode version is similar to the command line version except that there is no graphical output which allows it to be run in the background. All commands are stored in a journal file regardless of which version is used. These commands may then be replayed in the command line, batch mode or GUI versions.

For information on the commands and options used to execute CUBIT, see “Execution Command Syntax” on page 19.

Command Line Version

The command line interface provides the user access to all CUBIT commands via keyboard entry. When the command line version is executed, the command prompt (CUBIT>) appears in

the UNIX shell window or terminal. A graphics window pops up once a display-related command is executed.

The CUBIT command line interface allows the execution of any command at any time. In contrast, the GUI presents context driven hierarchical menus. The absence of a strict hierarchy enables the command parser to recognize any command entered at the CUBIT prompt. Commands may be abbreviated as long as they remain unique from other commands.

The command line interface provides an EMACS-style line editing input package with command history¹. It allows the user to edit the current line and move through the history list of lines previously typed. Commands replayed from a journal file are not saved in the history list. The available editing commands are defined in Table 3-1.

Table 3-1 Command Line Interface Line Editing Keys

Key ^a	Function
^A, ^E	Move to beginning or end of line, respectively
^F, ^B	Move forward or backward one position in the current line.
^D	Delete the character under the cursor. Sends end-of-file if no characters on the current line.
^H, DEL ^b	Delete the character to the left of the cursor.
^K	Delete from the current cursor position to the end of the line
^P, ^N	Move to the previous or next line in the history buffer.
^L	Redraw the current line.
^U	Delete the entire line.
NL, CR ^c	Places current input on the history list, appends a newline and returns that line to the CUBIT program for parsing.
?	Provides “instant” help. If no text has been entered prior to typing the ‘?’, a list of all valid keywords will be echoed. If some text has been entered, either a list of all valid keywords matching the entered text is echoed, or if the entered text only matches a single keyword, the syntax for that keyword will be echoed. If the ‘?’ is entered inside the single or double quotes of a filename, all files (with the correct suffix) matching the entered text will be echoed. The “correct” suffixes are .sat for solid models, .jou for journal files, .fsq for FASTQ files, .ps for hardcopy files, and .g for mesh files.

a. The notation ^X refers to holding down the control key and then typing the letter X. Case is not significant.

b. See the documentation for your keyboard/workstation to determine which key sends the DEL character.

c. NL is a newline, typically ^J, CR is a carriage return entered the normal way you end a line of text.

1. The command line interface package used in CUBIT is Copyright 1991 by Chris Thewalt. The following copyright notice appears in the source code: “Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that the above copyright notices appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation. This software is provided “as is” without express or implied warranty”.

Batch Interface

The CUBIT environment is available as a separate executable, typically called **cubitb**, that contains no graphical display capability. This implementation will operate as any other version of CUBIT, but is intended to perform unattended mesh generation. The batch implementation of CUBIT is invoked by entering '**cubitb -batch <journal_file>**' at the UNIX prompt¹. To initiate unattended operation, a journal file playback must be started. The journal file should contain the **Export** command to enable CUBIT to write out the model to a Genesis database file. Any graphics commands issued during a batch run are ignored. The EMACS-style line editing input package described in the previous section is also available in the batch version.

▼ Session Control

Several commands are available to control the overall CUBIT environment. Most of these commands are available in the GUI version by selecting the appropriate menu items in the **File** menu of the Main GUI window (Figure 3-1)

- **Exit.** The CUBIT session can be discontinued with either of the following commands
Exit
Quit
- **Reset.** A reset of CUBIT will clear the CUBIT database of the current geometry and mesh model, essentially allowing the user to begin a new session without exiting CUBIT. This is accomplished with the command

Reset.

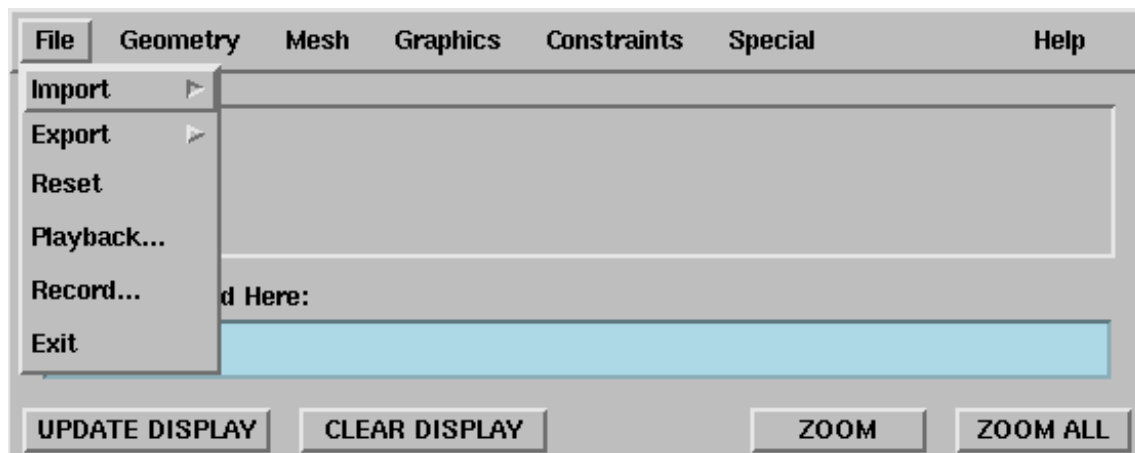


Figure 3-1 Main GUI Window Showing File Menu

General Execution Commands

CUBIT contains a few commands which control the executable in general. To determine the software version number, execute the **version** command which reports the version number, the date and time this version of CUBIT was compiled, the version number of the ACIS solid

1. See "Executing CUBIT" on page 19 for more information.

modeler, and the version number of the HOOPS library. This information is useful when discussing available capabilities or software problems with CUBIT developers.

Command echoing is controlled with the **[set] echo {on | off}** command. By default, commands entered by the user will be echoed to the terminal. The command **[set] logging {on | off} file 'filename'** can be used to additionally log all information output by CUBIT, including the command echo, to the file specified by **filename**.

▼ Journal Files

Journal files are used as a means to control CUBIT from simple text files and as insurance against lost work during execution. These files are created in various ways within CUBIT, or can be generated by any ASCII text editor by the user. They also serve as a means of error logging and user support. If a bug occurs and a journal file was being written, in many cases a support person can reproduce the error by simply playing the journal file.

CUBIT Journal File Generation

The CUBIT journalling facility can record all commands entered during the current CUBIT modeling session. By default, the journalling facility is on—if no journalling is desired, the user may issue the command **[set] Journal Off** or run CUBIT with the **-nojournal** command line option. Turning journalling off should be done with care, as the journal file can save model re-creation time when errors occur during a long session.

Unless turned off, the journal file is automatically created in the current directory. Most commands entered during the current modeling session are saved. The exceptions are commands that require interactive input (**mouse**, **pick**, **zoom cursor**), and the **play** command. The name of the journal file begins with the word “cubit” followed by a number between 01 and 99 followed by the characters “.jou”, for example, **cubit05.jou** or **cubit45.jou**. The number following “cubit” will increment as more journal files are generated in that directory.

In addition to the default journalling, specific portions of the CUBIT session can be saved to user assigned files. If running the GUI version, selecting the **Journal Record/Play** menu item from the **Special** menu will display the dialog box shown in Figure 3-2

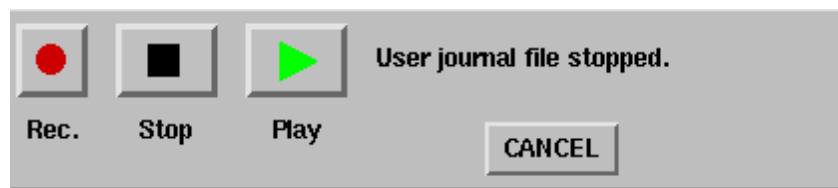


Figure 3-2 Journal Record/Play Dialog Box

To start recording a journal file, click the **Record** button. A File Selection dialog box will prompt the user for a filename. A typical file selection dialog box is shown in Figure 3-3. Recording can also be accomplished with the command

Record '<filename>'

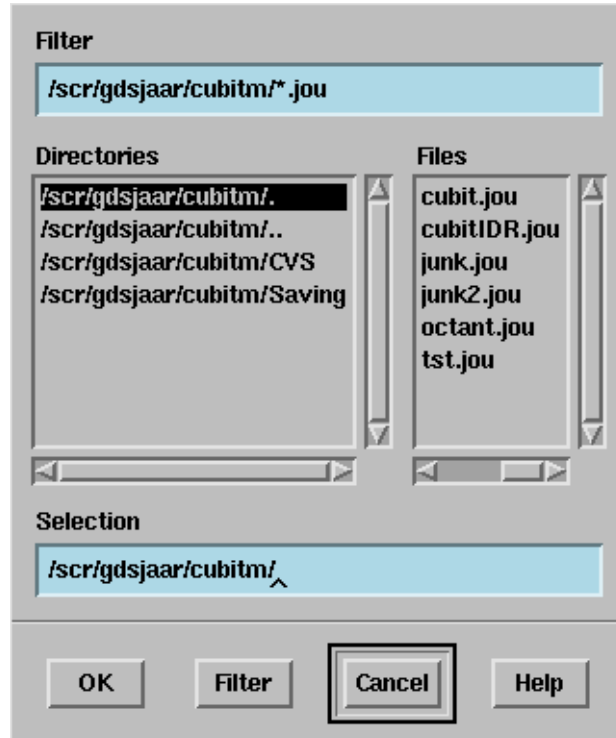


Figure 3-3 File Selection Dialog Box

Once initiated, all commands issued in CUBIT are copied to this file, as well as to the default journal files (if on). This journal file can be closed and recording to this file terminated by either pushing the Stop button shown in Figure 3-2, or with the command

Record Stop

The record command is particularly useful when a new finite element model is being built and alternate meshing strategies are being experimented with. Once the geometry has been defined, the record option can be used to record initial meshing controls and subsequent meshing commands. The mesh can be deleted, the recording terminated, and the process repeated to test alternate meshing strategies. To compare trial results, the user need only delete the current mesh and replay the journal file of the trial being considered.

Replaying Journal Files

In the GUI version, selecting the **Playback** menu item from the **File** menu, or clicking on the **Play** button of the dialog box shown in Figure 3-2 will display a File Selection dialog box that allows you to select a journal file name. To replay a journal file using the command line, issue the command

Playback '<filename>'

The file will be read and commands in the file executed. **Pause** commands can be inserted in the journal file to cause the command execution to pause at that point. Typing a return if running the command line version or clicking on the continue button in the pause dialog box will continue execution. Playback commands can be nested. Note that the filename must be enclosed in single quotes.

▼ Graphics

The graphics display window displays a graphical representation of the geometry and/or the mesh. This display is used in either the command line version or the GUI version of CUBIT. The quality and speed of rendering the graphics, the visibility, location and orientation of objects in the window, and the labeling of entities can all be controlled by the user. Additionally, multiple windows can be generated to provide multiple views.

The geometric model can be viewed from any point and shaded before mesh generation has occurred. The shaded representation of the part does not represent the actual surface of the geometry, but a faceted approximation of it computed by the ACIS® solid modeler. The wireframe representation only shows edges of the model, and can be misleading for geometry that has no edges (i.e. a sphere).

Similar to the geometric model, the mesh model can be viewed from any angle and displayed in either wireframe, hiddenline, polygonfill, flatshaded, painters or smoothshaded modes. The shaded representation of the part is generated directly from the quadrilateral faces which exist after the meshing process.

This section will discuss: 1) the control of the graphics window(s), 2) the control of the rendering parameters which affect the type, quality and/or speed of rendering of the image, 3) the control of which objects to draw and the color of drawn objects, 4) the desired labeling of objects on the image, 5) obtaining hard copy (e.g. postscript files) of the image, and 6) video animation generation.

As a general help to this section, when running the GUI version of CUBIT, most of the graphics controls are available under the **Graphics** menu item in the main window. To update the screen, a **Display** command must be issued.

Graphics Window Control

The graphics window is where the meshing graphics will be displayed and is the default viewport. The following attributes of the window can be controlled:

- **WindowSize**. The graphics window may be resized with the mouse, or with the commands

Graphics WindowSize Maximum

Graphics WindowSize <x_dimension> <y_dimension>

where **Maximum** will make the graphics window as big as the screen and the **x_dimension** and **y_dimension** are given in screen coordinates (pixels).

- **Background Color**. The window background color defaults to black but can be changed at any time using the command

Color Background <color_name>

Color Background <color_number>

where **color_name** is one of the colors listed in Appendix E, and **color_number** is an integer ID identifying the color. The background color can also be set using the **Graphics** menu **Color** dialog box as explained in section “Color” on page 53.

Multiple graphics windows can be created if desired. If multiple graphics windows are generated, only one of the display windows is active at any point in time -- that is, user

commands which affect the graphics display only affect the currently active window. The following user capabilities are provided:

- **Create Window.** Create a new window using the command

Graphics Window Create <window_id>

where **window_id** is an integer graphics window identifier in the range 1-9 (window 0 always exists). When a window is created, it is initialized to the default graphics state.

- **Delete Window.** Delete an existing window (note: window 0 cannot be deleted) using the command

Graphics Window Delete <window_id>

where **window_id** is the integer graphics window identifier of the window to be deleted.

Set Current Active Window. Select the graphics window which is to be currently active using the command

Graphics Window Active <window_id>

where **window_id** is the integer graphics window identifier of the selected graphics window (range 0-9).

Image Rendering Control

The type, quality, and speed of the rendered image can be controlled in CUBIT using several graphics mode types and rendering options. The **Graphics** menu **Mode** dialog box shown in Figure 3-4 is used to set these options in the GUI version of CUBIT. The graphics mode type is

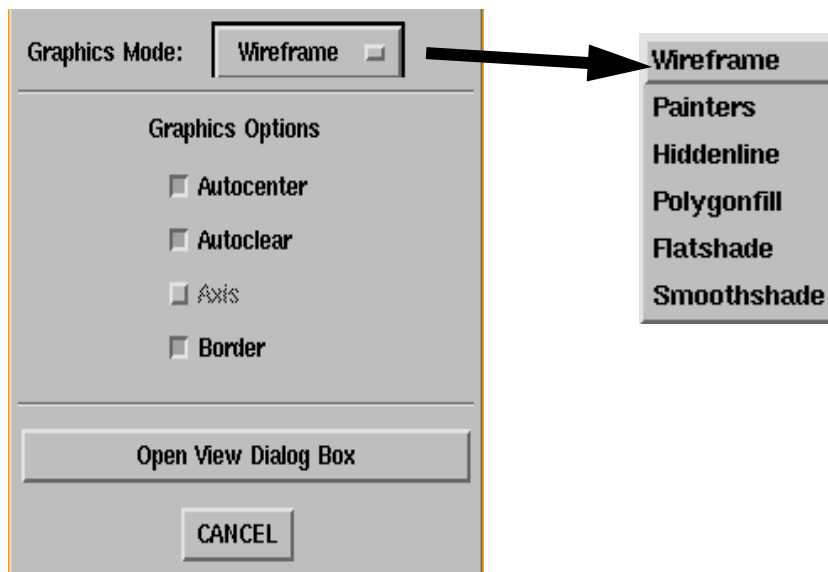


Figure 3-4 Graphics Mode Dialog Box

set by using the **Graphics Mode Type** option menu. The available graphics mode types are:

- **Wireframe.** Wireframe drawing is the quickest mode, but it also can be the most confusing if the mesh or the geometry is very complex. No hiddenline processing is done. The command to set this mode is

Graphics Mode Wireframe

- **HiddenLine.** This option produces an accurate hiddenline representation of the mesh or geometry. The command to set this mode is

Graphics Mode HiddenLine

- **PolygonFill.** This option is similar to flat shading, but uses a slightly different algorithm. The command to set this mode is

Graphics Mode PolygonFill

- **Painters**¹. This option produces a shaded image where each polygon is drawn in a single shade. The polygons are drawn in a depth-sorted order. Although a correct rendering is produced for most images, there are cases where an incorrect image may be rendered. This mode is usually faster than the FlatShade and SmoothShade modes. The command to set this mode is

Graphics Mode Painters

- **FlatShade.** This option produces images where each polygon is drawn in a single shade. The image is slightly degraded with this option, but the speed of rendering is improved. The command to set this mode is

Graphics Mode FlatShade

- **SmoothShade.** A smoothshaded image produces the highest quality picture, but at the most expense. Colors are blended continuously over the drawn surfaces. The command to set this mode is

Graphics Mode SmoothShade

- **Dual.** This mode is designed to show the dual of the mesh as generated during whisker weaving. With this setting, the outside element edges are drawn as wireframe and the whisker sheets are drawn in smooth shading. This allows for a nice image of the structure of the dual of a hexahedral mesh. The command to set this mode is

Graphics Mode Dual

The **Graphics** menu **Mode** dialog box shown in Figure 3-4 is also used to set the graphics mode options. These options control details of how the image is controlled between displays, and the type of enhancements added to the regular drawing modes. All options default to **On** at the start of execution. The graphics mode options are chosen by pushing the appropriate radio buttons in the **Graphics Mode Options** region of this window. The graphics mode options available in CUBIT are:

- **Autocenter.** This option automatically centers the model in the viewport. The command to set this option is

Graphics Autocenter {On | Off}

- **Autoclear.** This option automatically clears the graphics window between displays, or updates. The command to set this option is

Graphics Autoclear {On | Off}

- **Border.** This option draws a border around the current viewport. The command to set this option is

Graphics Border {On | Off}

1. The terminology “painters” is used since it draws the scene similar to the method used by a painter who might paint closer objects over more distant objects.

- **Axis.** This option controls the display of the axis or coordinate triad. The command to set this option is

Graphics Axis {On | Off}

- **LineWidth.** This option controls the width of the lines used in the wireframe and hiddenline displays. The command to set the line width is

Graphics LineWidth <width>

- **Text Size.** This option controls the size of text drawn in the graphics window. The size given in this command is the desired size relative to the default size. The command to set the text size is

Graphics Text [Size] <size>

All option settings take affect at the time they are selected, it is not necessary to apply the changes. The **Set View Parameters** is a short-cut method for popping-up the Graphics View Dialog Box described in the next section.

Two additional commands,

graphics clear

graphics center

are available from the command line to perform a one-time only clear of the graphics window or centering of the model in the viewport. They do not affect the setting of the autoclear and autocenter toggles.

Viewing the Model

Figure 3-5.shows a schematic of the variables that effect the view of the object. Adjusting these

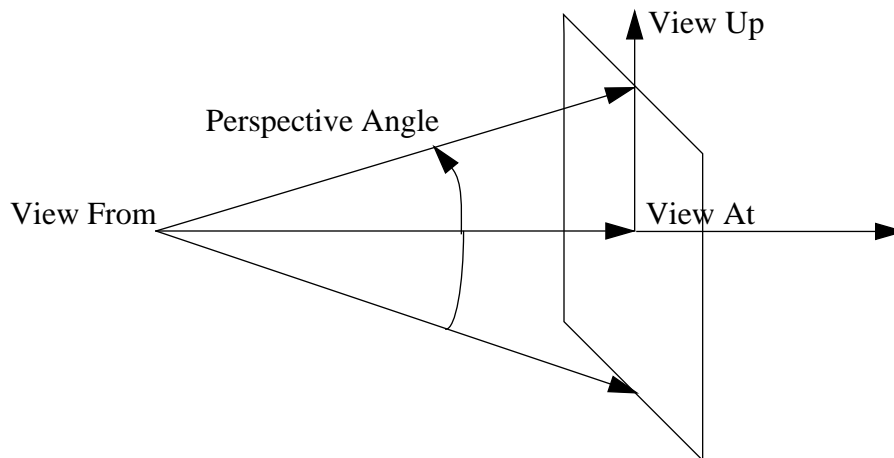


Figure 3-5 Schematic of From, At, Up, and Perspective Angle

variables will effect the way the three-dimensional model is projected onto the two-dimensional screen. These adjustments require you to update the display to see the results. To change the view parameters in the GUI version, select the **View** menu item from the **Graphics** menu. The dialog box shown in Figure 3-6 will be displayed. This dialog box will show the current values

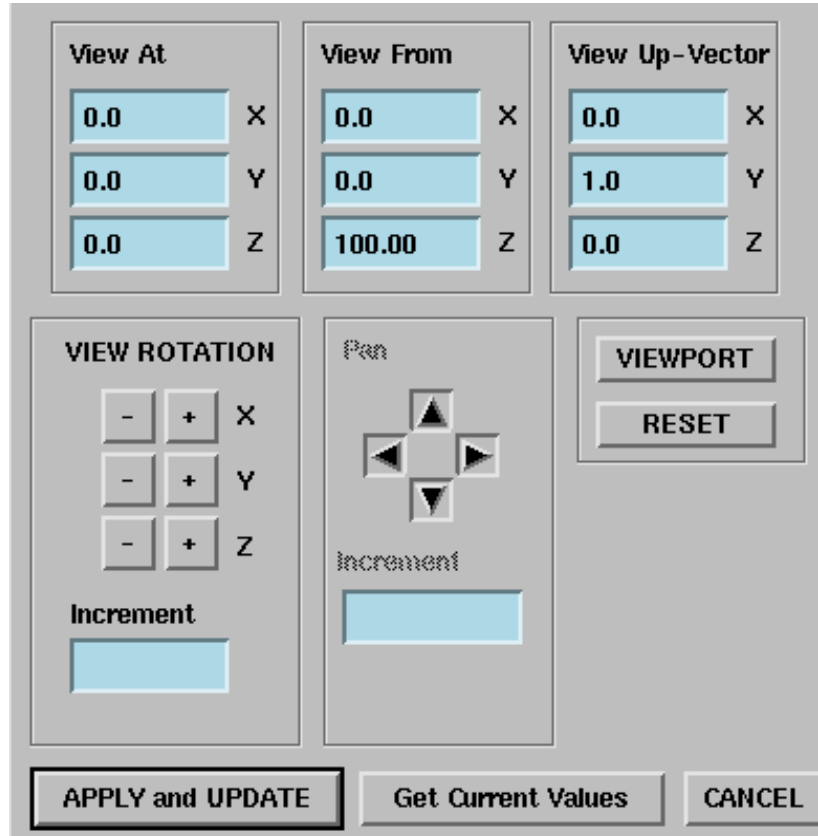


Figure 3-6 The Graphics View Dialog Box

for the ‘at’ point, the ‘from’ point and the up vector. The following adjustments can be made by the user:

- **View At Point.** The point you are viewing or looking ‘at’ can be set using the **X**, **Y**, and **Z** coordinates of the **View ‘At’** text fields. To set the looking ‘at’ point using the command line, issue the command

[View] At <x> <y> <z>

- **View From Vector.** The point you are viewing ‘from’ can be set using the **X**, **Y**, and **Z** coordinates of the **View ‘From’** text fields. If automatic centering (see “Image Rendering Control” on page 45) is on, the input ‘from’ vector defines a relative viewpoint away from the ‘at’ point, in the direction of the ‘from’ vector. The magnitude of the ‘from’ vector is computed so that the picture fits nicely on the screen. When automatic centering is off, the ‘from’ vector defines an absolute viewpoint. To set the viewing ‘from’ vector using the command line, issue the command

[View] From <x> <y> <z>

- **Up Vector.** The up vector sets the orientation for the graphical display. In other words, along the line which connects the ‘from’ and the ‘at’ point, the up vector specifies which direction is displayed as up on the screen. This can be set using the **X**, **Y**, and **Z** coordinates of the **Up Vector** text fields in Figure 3-6 or using the command

[View] Up <x> <y> <z>

- **Rotate.** The rotation of the view can be specified by an angle about a world axis, or about a screen axis vector positioned at the focus point (Screen) or the camera. Additionally rotations

can be specified about any general axis by specifying start and end points to define the general vector. The *right hand rule* is used in all rotations. To rotate about an axis using the GUI interface, place a rotation angle (in degrees) in the **Increment** text field shown in Figure 3-6. Rotation is performed by clicking on either the **+** or **-** button by the desired axis (**X**, **Y**, or **Z**). The command to accomplish such a rotation is

**[Rotate <angle> About [Screen | World | Camera] {X | Y | Z}
[Animation Steps <number>]**

The command defaults to the Screen coordinate system (rotations about a screen axis that is translated to the focus point). The Animation Steps option is included in this command (and all other rotation commands) to allow the user to perform a “smooth” rotation using several steps. This will let the picture appear to start and stop the total rotation *smoothly*. This is particularly useful when producing a video animation sequence (“Video Animations” on page 55) where a smooth sequence is desired. If the video system has been initialized, the animation command will take a snapshot at each step of the rotation.

Continuous rotations about any axis can be performed by double clicking one of the **+** or **-** buttons. The view is changed in sequential steps by the rotation increment specified. The screen is updated as fast as the picture can be processed. When the desired view is obtained the continuous rotations can be stopped by clicking the mouse somewhere in the View dialog box. These continuous rotations are not actually sent as commands to the parser, and as such are not stored in the journal file. To save the final state of the viewing angle, push the **Get Current Values** button in Figure 3-6 followed by the **Apply and Update** button. This will store the viewing parameters in the journal file. Continuous rotations are not available in the command line version.

Rotations can also be performed about the line joining the two vertices of a curve in the model, or a line connecting two vertices in the model. This is done with the commands¹

**Rotate <angle> About Curve <curve>
[Animation Steps <number>]**

**Rotate <angle> About Vertex <vertex_1> Vertex <vertex_2>
[Animation Steps <number>]**

- **Perspective.** The perspective angle can be set to adjust the relative perspective distortion of the view. A value of 0.0 will produce no distortion as if the viewing “from” location was at infinity. A larger value will produce more distortion. Values of about 15.0 degrees are normal. The perspective angle is set using the command

Graphics Perspective Angle <angle>

A more convenient method of adjusting the perspective is with a simple on/off toggle. This toggles the perspective angle between 0.0 and the current setting. The command to toggle perspective on and off is

Graphics Perspective {On | Off}

- **Zooming.** The image can be zoomed to provide a close-up view of portions of the image. When using the GUI, the **Zoom** and **Zoom Reset** buttons on the main GUI window (Figure 3-1) are used. The **Zoom** button allows a zoom window to be defined on the screen using the mouse. When the **Zoom** button is clicked the cursor will move to the Graphics Window and become a cross-hair cursor. Click the left mouse button and drag the cursor over the desired zoom area and then release the mouse button. The third mouse button will cancel a zoom if

¹1. See “Geometry Definition” on page 67 for definitions of Curve and Vertex

clicked before the user clicks the first mouse button.

The command line version provides similar functionality with the

[Graphics] Zoom Cursor

command. After entering the command, move the cursor to the graphics window and click the left mouse button at both corners of the desired zoom area. The command for performing a zoom is

[Graphics] Zoom <x_min> <y_min> <x_max> <y_max>

where the values specified are in screen coordinates (between 0 & 1). The **Zoom Reset** button updates the zoom limits to a size appropriate to capture all currently defined entities. The equivalent command is

[Graphics] Zoom Reset

A simplified command line version of zoom has also been implemented that takes a single argument. The command syntax is defined as

[Graphics] Zoom Screen <scale_factor>

where `scale_factor` scales the view distance. Values of `scale_factor` > 1.0 zoom in toward the focus point while values of `scale_factor` < 1.0 zoom away out from the focus location. The best way to think of this is to think of a magnifying glass, a 2x zoom would magnify an object. This command is not supported in the GUI.

The **Get Current Values** button in Figure 3-6 displays the current ‘at,’ ‘from,’ and ‘up vector’ in the appropriate text fields. This is useful after doing multiple rotations. To journal a viewing angle after doing a continuous rotate command, select this button then click the **Apply and Update** button. The commands

List View

View List

will list the current values of the At point, From point, Up vector, and Perspective angle in the command line version.

Displaying Entities

The entities to be drawn in the current image can be controlled to limit the amount of information presented on the screen. There are two distinct modes of getting entities to the screen. The first is to set the visibility of the entities desired to be on and to turn the rest off. This visibility control establishes a “display list” of items that will be included in the image every time it is redrawn with a **display** command. The second method is an *immediate mode* drawing capability. Using a number of **draw** commands, individual items can be drawn onto the current picture. A draw command does not put the designated entities on the “display list” - it simply draws their wire frame image over the top of the current image. This immediate mode drawing is useful in highlighting specific nodes, faces, etc., but will not change the picture that is displayed when the image is updated using the **display** command. This section uses geometry and mesh terms defined in “Geometry Definition” on page 67, “Mesh Definition” on page 89, and “Finite Element Model Definition” on page 123. The reader may want to read those sections prior to reading the following discussion.

Drawing Entities

The series of **Draw** commands allow inspection of individual geometric and mesh entities. Individual entities or ranges of entities can be displayed. The draw commands affect the graphics system only temporarily and updating the display will show only those items actually in the display list.

When using the GUI version, select the **Draw** menu item from the **Graphics** menu to display the dialog box show in Figure 3-7. Select an entity type to be drawn, enter an ID or a range of IDs for that entity type, and then click **Apply** to draw the entities.

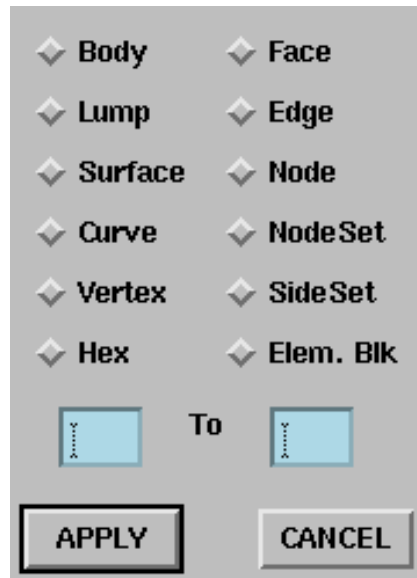


Figure 3-7 Graphics Draw Dialog Box

The command line equivalent to draw selective entities is:

```
draw {body | curve | edge | face | hex | volume |
      node | nodeset | sideset | surface | vertex | group} <id_range>
```

If **autoclear** mode is enabled, each **draw** command will clear the screen prior to updating the display. If **autoclear** mode is disabled, the specified entities will be added to the current set of displayed entities. An explicit **clear** command may be issued at any time to clear the display.

Highlighting Entities

An entity can be highlighted without erasing the remainder of the displayed model using the Highlight commands. These commands highlight the entity in the highlight color. The highlight commands available on the command line are:

```
highlight {body | curve | surface | vertex | volume} <id_range>
```

Currently, the highlight color is defaulted to a light gray.

Setting Visibility

Visibility of the geometry and the mesh is controlled by the use of global settings as well as through the use of individual (selective) geometric entity settings. In the GUI version, selecting the **Visibility** menu item from the **Graphics** menu will display the Visibility

dialog box shown in Figure 3-7. The **Global/Selective Mode** can be set to either **Glo-**

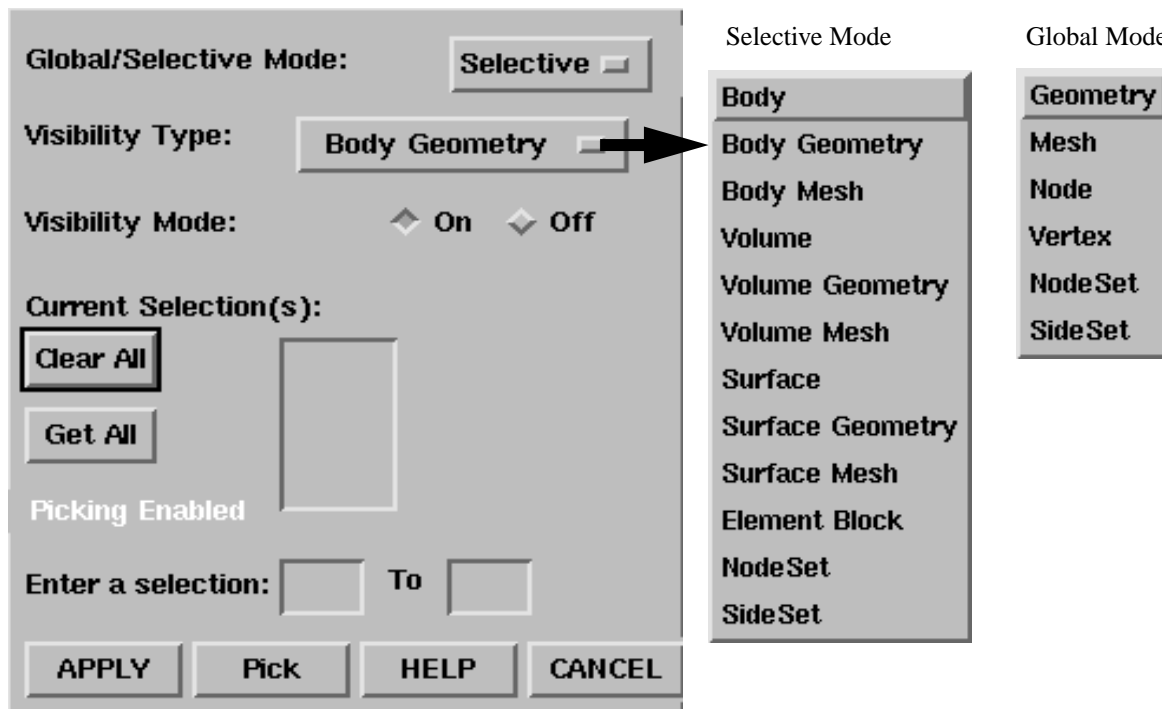


Figure 3-8 Visibility Dialog Box

bal or **Selective**. The **Visibility Type** will change to the proper list based on the mode. The **Visibility Mode** radio buttons control whether visibility is set to **on** or **off**.

• Global Settings

Choosing the **Global Mode** will change the **Visibility Type** to include the appropriate global items when using the GUI (See Figure 3-8). The following global settings can be used to adjust the display list:

- **Geometry**. This sets the visibility of all geometric entities.
- **Mesh**. This sets all mesh entities to be set to on or off.
- **Node**. This sets the drawing of mesh nodes (small dots).
- **Vertex**. This sets the drawing of geometric vertices (small dots).
- **NodeSet**. This sets the drawing of all nodes (small dots) in a NodeSet to be either on or off.
- **SideSet**. This sets the drawing of all element faces (or edges) in a SideSet to be either on or off.

The command line equivalent for setting global visibility flags is:

```
{block | geometry | mesh | node | vertex | nodeset | sideset } visibility {on | off}
```

• Individual Geometric Entity Settings

Two visibility flags are attached to individual geometric entities: 1) a flag indicating whether the geometry itself is to be included in the display list (visible), and 2) a flag to indicate if the mesh attached to the geometry is to be visible. Choosing the **Selective Mode** will change the **Visibility Type** to include the appropriate selective items when using the GUI. For each geometric entity, the visibility of the item and any owned mesh can be set, or just the geometry visibility or the mesh visibility can be set. The visibility for bodies, volumes and surfaces can also be set with this interface.

The command line equivalents for the selective visibility flags are:

```
{body | volume | surface | sheet | group} <range> visibility {on | off}
{body | volume | surface | group} <range> geometry visibility {on | off}
{body | volume | surface | group} <range> mesh visibility {on | off}
```

Color

The **Color** commands give the user customization control of the screen appearance of any geometric entity and its owned mesh entities. The default color used for an entity is the color of the owning entity¹. For example, if the color of a curve is not specifically set, it inherits the color of the owning surface. Mesh entity colors are determined by the owning geometry entity, unless set specifically according to the nodeset, sideset, or mesh entity color commands. The user can also control the color of the screen background.

The colors available at this time are listed in Appendix E. To change the color of an entity in the GUI version select the **Color** menu item from the **Graphics** menu. A Color dialog box will be displayed (see Figure 3-9) from which the geometry type of the entity to be changed can be selected. Choose a color from the list. The color chosen will be displayed in the swatch area to the right of the list. Pick the entities to be modified and click **Apply**.

The command line equivalents to change colors are:

```
color {body | volume | surface | nodeset | sideset | block | sheet | group}
    <id_range> <color name>
color {body | volume | surface | group} <id_range> mesh <color name>
color {body | volume | surface | group} <id_range> geometry <color name>
color {node | background} <color_name>
```

Entity Labeling

All geometric entities can be labelled with unique (to their geometric type) labels and, optionally the number of mesh intervals assigned to the entity, to enable specific entity identification. All mesh entities can also be labelled with unique (to their mesh entity type) labels to enable specific entity identification. In addition, the names of the geometric entities can also be used for the label. If the The labels are turned on or off by using the **Label** commands.

1. See “Geometric Topology” on page 67 and “Mesh Definition” on page 89 for a description of the ownership of geometric and mesh entities.

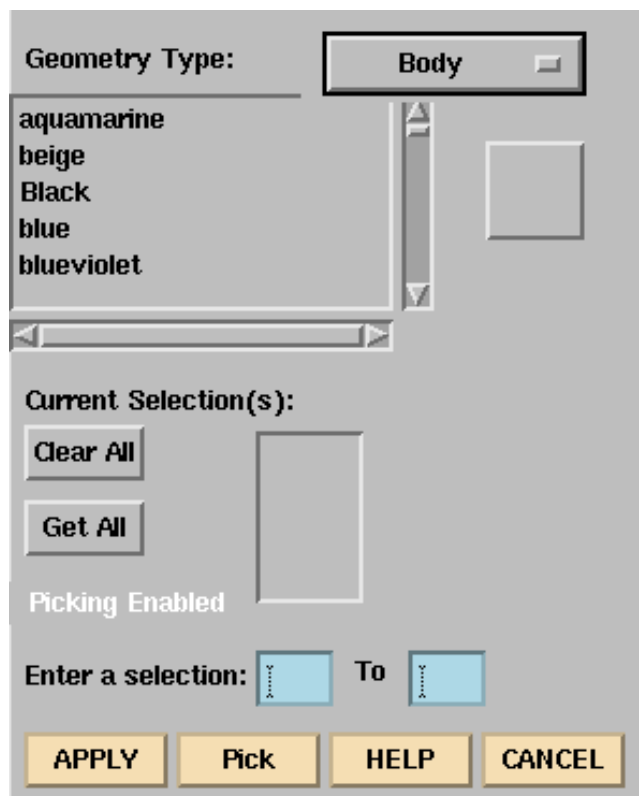


Figure 3-9 Color Dialog Box

The labels will be displayed on the entity's centroid, which is helpful in the screen picking operations which are used in the GUI version of CUBIT. The screen picks try to locate the entity with the closest centroid to the actual screen pick. Thus by turning entity labelling on, the user knows exactly where to click the pointing device in order to pick a specific entity. Labels are also useful in determining which entities were merged during a Feature Consolidation operation.

The command line commands are:

label all {on | off}

label geometry {on | off}

label mesh {on | off}

label {body | volume | surface | curve | vertex} {on|off|name|interval|id}

Label {face | edge | hex | node} {on | off}

In addition, the size of text drawn to the graphics display, which includes entity labels, can be adjusted with the **Graphics Text Size** command; the command line syntax for this command is:

Graphics Text Size <size_factor>

where **<size_factor>** is a scaling factor relative to the default text size.

Hardcopy Output

The **hardcopy** command is used to capture graphics output to a PostScript or PICT file. Color or monochrome PostScript and encapsulated PostScript files are available.

In the GUI version, selecting the **Hardcopy** menu item from the **Graphics** menu displays the dialog box shown in Figure 3-10. This dialog box is used to define PostScript capture options. Input a filename to be used for the PostScript capture in the filename text field. Select a radio button to define either encapsulated PostScript (**EPS**) or normal **PostScript** and a **radio** button to define either **Color** or **Monochrome** PostScript. Click the **Apply** button to capture the current display or **Cancel** to abort the output. Only one display can be written to a file at the current time. A new filename must be specified for each display or the file will be overwritten. The output defaults to non-encapsulated color PostScript.

The command line equivalent is:

hardcopy '**<filename>**' [**encapsulated|postscript|eps**] [**color|monochrome**]

Hardcopy '**<filename>**' pict [**xsize <xpixels>**] [**ysize <ypixels>**]

where xsize and ysize specify the pixel size of the created PICT image.

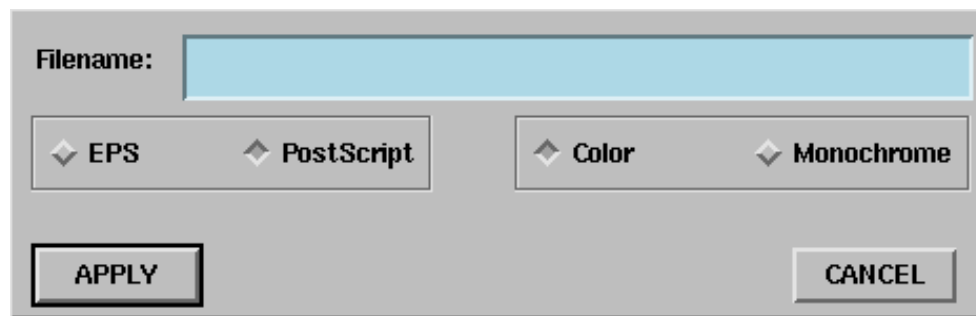


Figure 3-10 Hardcopy Output Dialog Box

Video Animations



Several commands are available for generating a video recording of the graphics on the screen. The actual video initialization and recording has been set to work with the system in the graphics lab of the computational mechanics department. To initialize the video system, the commands

Video Initialize [Frames <frames>]

Video Initialize '**<base_filename>**' pict [**xsize <xsize>**] [**ysize <ysize>**]

are used. The first command will set up the recording devices, move the recorder to the first available frame, and set the system ready to record the picture displayed on the workstation screen next to the video recording equipment. The disk will be initialized to the specified number of frames or 2000 if not specified. It has been found that resizing the window with the following commands positions the graphics display in the proper position for optimal recording:

Graphics WindowSize Maximum

Graphics WindowSize 1170 820

The second command is used to create a separate PICT file of each frame. These PICT files can then be converted into a QuickTime or MPEG animation file using external software. The `base_filename` will have the frame number appended to the end to specify the sequence. The default PICT size is 640 pixels horizontal by 480 pixels vertical.

The video animation can now be generated by taking sequential snapshots of the screen. Each snapshot is captured by issuing the command:

Video Snap

Many of the meshing algorithms have been imbedded with flags that allow incremental snaps during the meshing process. These flags are activated by the initialization of the video device. This allows the generation of video animations of the meshing process rather easily. Often it is useful during an animation sequence to spin the picture around for the viewer to see the geometry and/or the mesh. These spins will appear choppy on the resulting video unless the object starts and stops rotation *smoothly*. The animated rotations described in the rotation section on page 48 is designed to provide this *smooth* rotation effect. These commands are also useful for showing the model to others during demonstrations. For video animations, 30 to 60 steps are needed when making a full revolution. If the video system has been initialized, the animation command will take a snapshot at each step of the rotation.

▼ Model Information

Information about the current CUBIT model can be obtained through the **List** commands. There are five general areas for which this information can be obtained: Model Summary, Geometry, Mesh, Special Entities, and Other. These are described in detail below. The descriptions will include sample output for some of the commands. To provide a frame of reference, the output will be for a 1x2x3 cube meshed with an average element size of 0.1 specified for the body. The journal file used to create the model is shown in Table 3-1.

Table 3-1 CUBIT Journal file used for List Output Examples

```
brick x 1 y 2 z 3
body 1 size 0.1
mesh volume 1
block 1 volume 1
nodeset 1 surface 1
sideset 1 surface 2
group "Surfaces" add surface 1 to 6
```

Model Summary Information

A brief summary of the current state of the model can be obtained from the list totals or list model command. The same information results from either command and provides information on the number of each type of geometric, mesh, and special entity in the current model. A sample output is shown in Table 3-2.

Table 3-2 Sample Output from ‘List Model’ Command

```
CUBIT> list model

Model Entity totals:
  Geometric Entities:
    Total groups:          1
    Total bodies:          1
    Total volumes:         1
    Total surfaces:        6
    Total curves:         12
    Total vertices:        8

  Mesh Entities:
    Total Hex elements:    6000
    Total mesh faces:      7876
    Total mesh edges:      9854
    Total mesh nodes:      7161

  Special Entities:
    Total Element Blocks:  1
    Total SideSets:        1
    Total Nodesets:        1
    Total BoundaryLayers:  0
```

Geometry Information

The commands related to listing information about the geometry of the model are

list names [group|body|volume|surface|curve|vertex|all]

**list {group | body | volume | surface | curve | vertex} <id_range>
[geometry] [debug]**

The first command will list names currently used in the model and their corresponding entity type and id. If the all identifier is specified, all names in the model will be listed; if one of the other identifiers is entered, only names for that specific entity type will be output. Sample output from the list names surface command is shown in Table 3-3. This output shows that, for example, Surface 2 has the two names ‘BackSurface’ and ‘Surface2’..

Other information that can be determined from this output is that there are six surfaces in the model and the IDs of these surfaces range from 1 to 6. For larger models in which several geometry decomposition operations have performed, this information is sometimes very useful for determining the entities and their ids that are in the current model.

The second geometry-related information command provides more detailed information for each of the specific entities. This information will include the name and id of the entity, its meshed status, the number of owned mesh entities (if it is meshed), the settings for various meshing-related parameters (scheme, smooth scheme, size, and number of intervals), and a summary of the next lower-dimension geometry entities that make up this entity. The optional **geometry** identifier will additionally list the geometric bounding box for the entity. The optional **debug** identifier provides additional solid model information that is typically only of interest to developers of the geometry-related code. Table 3-4 through Table 3-9 show sample output for each of the group, body, volume, surface, curve, and vertex listing options.

Table 3-3 Sample Output from ‘List Names’ Command

```
CUBIT> list name surf
_____Name_____ Entity Id
BackSurface      Surface 2
BottomSurface    Surface 3
FrontSurface     Surface 1
LeftSurface      Surface 4
RightSurface     Surface 6
Surface1         Surface 1
Surface2         Surface 2
Surface3         Surface 3
Surface4         Surface 4
Surface5         Surface 5
Surface6         Surface 6
TopSurface       Surface 5
```

Table 3-4 Sample Output from ‘List Group’ Command

```
Group Entity 'Surfaces' (Id = 1)
Owned Entities:
_____Name_____ _____Entity_____ Scheme/Meshed Int Int Size
FrontSurface Surface 1 map/Y 1 0.100000
BackSurface Surface 2 map/Y 1 0.100000
BottomSurface Surface 3 map/Y 1 0.100000
LeftSurface Surface 4 map/Y 1 0.100000
TopSurface Surface 5 map/Y 1 0.100000
RightSurface Surface 6 map/Y 1 0.100000
```

Table 3-5 Sample Output from ‘List Body’ Command

```
CUBIT> list body 1
Body Entity 'Body1' (Id = 1)
Owned Volumes:
_____Name_____ Id: Meshed: Use Count:
Volume1 1 Yes 1
```

Mesh Information

The command related to listing information about the model mesh is:

list { hex | face | edge | node } <id_range>

The output from this command provides detailed information about the specified entities. This information will include the id of the entity, its owning geometry entity, and other entity-specific information. The hex output will indicate the Exodus Id¹, the volume which owns the hex element, and the eight corner nodes (in standard Exodus order). The face output lists the volume or surface which owns the mesh face, its four corner nodes, and a list of hexes that possibly share this face. The edge output lists the volume, surface, or curve which owns the mesh edge, its two

1. The id of the hex when written to the Exodus database, not the CUBIT id. The default value is -1 before writing the Exodus database.

Table 3-6 Sample Output from ‘List Volume’ Command

```
CUBIT> list volume 1
Volume Entity 'Volume1' (Id = 1)

    Meshed:          Yes
    Mesh Scheme:     map
    Smooth Scheme:    equipotential

Interval Count: 1
Interval Size:  0.100000
Block Id:       1
```

Owned Surfaces:		Mesh Scheme/		Interval:		
_____Name_____	Id	Meshed	Smooth Scheme	#	Size	
Surface1	1	map/Y	equipotential	1	0.100000	
Surface2	2	map/Y	equipotential	1	0.100000	
Surface3	3	map/Y	equipotential	1	0.100000	
Surface4	4	map/Y	equipotential	1	0.100000	
Surface5	5	map/Y	equipotential	1	0.100000	
Surface6	6	map/Y	equipotential	1	0.100000	

Table 3-7 Sample Output from ‘List Surface’ Command

```
CUBIT> list surf 1
Surface Entity 'FrontSurface' (Id = 1)

    Meshed:          Yes
    Total element faces:          200
    Total nodes (all inclusive):  231
    Mesh Scheme:      map
    Smooth Scheme:    equipotential

    Interval Count: 1
    Interval Size:   0.100000
    Block Id:        0

    Total number of curves:  4
```

_____Name_____	Id	Scheme/ Meshed	Length	Interval: Number	Size	Factor	Vertices: Start,	End
Curve1	1	equal/Y		2	20 H	0.1	1/Y	2/Y
Curve2	2	equal/Y		1	10 H	0.1	2/Y	3/Y
Curve3	3	equal/Y		2	20 H	0.1	3/Y	4/Y
Curve4	4	equal/Y		1	10 H	0.1	4/Y	1/Y

end nodes, the length of the edge, and a list of faces that possibly share this edge. The node output lists the coordinates and the volume, surface, curve, or vertex that owns the node.

Table 3-10 through Table 3-13 show sample output for each of the hex, face, edge, and node options.

and

Table 3-8 Sample Output from ‘List Curve’ Command

CUBIT> list curve 1 to 12 by 2								
Name	Id	Scheme/		Interval:			Vertices:	
		Meshed	Length	Number	Size	Factor	Start,	End
Curve1	1	equal/Y		2	20 H	0.1	1/Y	2/Y
Curve3	3	equal/Y		2	20 H	0.1	3/Y	4/Y
Curve5	5	equal/Y		2	20 H	0.1	5/Y	6/Y
Curve7	7	equal/Y		2	20 H	0.1	7/Y	8/Y
Curve9	9	equal/Y		3	30 H	0.1	4/Y	7/Y
Curve11	11	equal/Y		3	30 H	0.1	3/Y	8/Y

Table 3-9 Sample Output from ‘List Vertex’ Command

CUBIT> list vertex 1 to 8						
Name	Id	Meshed	X-coord	Y-coord	Z-coord	
Vertex1	1	Yes	0.500000	-1.000000	1.500000	
Vertex2	2	Yes	0.500000	1.000000	1.500000	
Vertex3	3	Yes	-0.500000	1.000000	1.500000	
Vertex4	4	Yes	-0.500000	-1.000000	1.500000	
Vertex5	5	Yes	0.500000	1.000000	-1.500000	
Vertex6	6	Yes	0.500000	-1.000000	-1.500000	
Vertex7	7	Yes	-0.500000	-1.000000	-1.500000	
Vertex8	8	Yes	-0.500000	1.000000	-1.500000	

Table 3-10 Sample Output from ‘List Hex’ Command

CUBIT> list hex 1000 to 6000 by 1000							
Hex ID	Exodus ID	Owned By	Contains Nodes:				
1000	1000	Volume	1	2886	1358	28	126
				3057	1357	27	125
2000	2000	Volume	1	3741	1353	23	121
				3912	1352	22	120
3000	3000	Volume	1	4596	1348	18	116
				4767	1347	17	115
4000	4000	Volume	1	5451	1343	13	111
				5622	1342	12	110
5000	5000	Volume	1	6306	1338	8	106
				6477	1337	7	105
6000	6000	Volume	1	7161	1333	3	101
				691	690	1	82

Special Entity Information

Special entities include element blocks, sidesets and nodesets (boundary conditions), and boundary layers. Whisker weaving-specific information including whisker sheets and whisker hexes are also considered special entities. This information includes the number of mesh entities in the special entity and a list of the geometry entities owned by the special entity. Sample output for the list block, list sideset, and list nodeset commands are shown in Table 3-14, Table 3-15, and Table 3-16, respectively.

Table 3-11 Sample Output from ‘List Face’ Command

```
CUBIT> list face 1 to 7876 by 1000
```

Mesh Face	Owned By		Nodal Connectivity			Shared by Hexes:		
1	Surface	6	1	3	101	82	6000	
1001	Surface	3	662	1072	1101	682	5820	
2001	Surface	5	2009	2010	2039	2038	3921	
3001	Volume	1	2143	2412	2583	2142	441	461
4001	Volume	1	1208	3646	3665	1237	1700	1900
5001	Volume	1	4577	1319	1318	4748	2960	2980
6001	Volume	1	5777	5776	602	573	4183	4383
7001	Volume	1	6638	6637	6808	6809	5389	

Table 3-12 Sample Output from ‘List Edge’ Command

```
CUBIT> list edge 1 to 9854 by 1000
```

Edge	Owned By		Start/End Node		Length	Shared by Faces:		
1	Curve	10	1	3	0.100000	1	1271	
1001	Surface	6	543	542	0.100000	458	488	6331
2001	Surface	2	1047	1046	0.100000	954	974	2300
3001	Surface	4	1543	1542	0.100000	1458	1488	6230
4001	Surface	5	1992	2021	0.100000	1982	1983	3852
5001	Volume	1	2393	2222	0.100000	2830	2831	2836
6001	Volume	1	3551	3532	0.100000	3854	3855	4018
7001	Volume	1	4589	319	0.100000	4879	4880	4883 5043
8001	Volume	1	5629	5458	0.100000	5905	5906	5909
9001	Volume	1	6668	6497	0.100000	6930	6931	6936

Table 3-13 Sample Output from ‘List Node’ Command

```
CUBIT> list node 1 to 7161 by 1000
```

Node	X-coord	Y-coord	Z-coord	Owner	
1	0.500000	-1.000000	1.500000	Vertex	1
1001	-0.100000	0.400000	-1.500000	Surface	2
2001	0.200000	1.000000	1.300000	Surface	5
3001	0.200000	0.900000	-1.000000	Volume	1
4001	0.000000	-0.300000	-0.400000	Volume	1
5001	-0.100000	0.400000	0.200000	Volume	1
6001	-0.300000	-0.800000	0.800000	Volume	1
7001	-0.400000	-0.100000	1.400000	Volume	1

Table 3-14 Sample Output from ‘List Block’ Command

```
CUBIT> list block 1
Block 1 contains 6000 3D element(s) of type HEX8.
Owned Entities:
Volume 1 Meshed
```

Other Information

Other non-geometry and non-mesh information is also provided through the list commands. These include message output settings, memory usage, and graphics settings.

Table 3-15 Sample Output from ‘List SideSet’ Command

```
CUBIT> list sideset 1
SideSet 1: contains 200 element sides.
Owned Entities:
Surface 2          Meshed
```

Table 3-16 Sample Output from ‘List NodeSet’ Command

```
CUBIT> list nodeset 1
NodeSet 1: contains 231 nodes.
Owned Entities:
Surface 1          Meshed
```

Message Output Settings

There are several types of messages output by CUBIT that are of interest to CUBIT users and developers. These messages and the type of information they convey are:

- **Information**- messages that contain information that is helpful to the user, but not critical to the operation of the program.
- **Warning** - messages that signal problems that may or may not be important to the operation of CUBIT.
- **Error** - messages signaling errors in the operation of CUBIT; these types of errors usually result in the termination of the program.
- **Debug** - debugging messages used by CUBIT developers.

The printing of Information, Warning and Debug messages can be turned on or off with the appropriate set command (Error messages are always printed). There are multiple Debug message flags, each controlling debug output for a different part of CUBIT. The value of each message flag or all message flags can be printed with the **List Settings** command. The commands used to print the value of message flags are:

list {echo | info | warning | journal | debug | settings}

Message flags can also be set using command line options; the Warning and Information flags are set with **-warning={on|off}** and **-information={on|off}** options, respectively. The Debug flags are set with **-debug=<setting>**, where **<setting>** is a comma-separated list of integers or ranges of integers. An integer range is specified by separating the beginning and the end of the range by a hyphen. For example, to set debug flags 1, 3, and 8 to 10 on, the syntax would be **-debug=1,3,8-10**. Flags not specified are off by default. Debug messages are typically of importance only to developers and are not normally used in normal execution.

The **List Settings** command lists the value of all the message flags, as well as the journal file and command echo settings; an example of the output from this command is shown in Table 3-17. The first several lines indicate the current settings of the debug flags, where the debug output will be output if the flag is on, and a short description of the purpose of the debug flag. For example, debug flag is enabled, its output will be written to the file ‘timing.log’ and the purpose of the flag is to output timing information.

Following the debug flag information is the settings of the echo, info, journal, and warning flags. Typically these should always be enabled. The final line of the output indicates whether logging is enabled and if so, where the information will be output. If logging is enabled, all echo, info,

warning, and error messages will be output both to the terminal and to the logging file. The other options to the list command select specific subsets of the list settings output.

The settings of the info, warning, journal, logging, and debug flags is set via the following commands:

```
[set] logging {on | off} file 'filename'
[set] {info | warning | journal} {on | off}
[set] debug <id> {on | off} terminal
[set] debug <id> {on | off} file 'filename'.
```

Table 3-17Sample Output from ‘List Settings’ Command

```
CUBIT> list settings
Debug Flag Settings (flag number, setting, output to, description):
 1  OFF  terminal          User Interface: If flag is enabled, filenames being
                           used for input will be echoed and each input line
                           will be echoed prior to being parsed.
 2  OFF  terminal          Whisker weaving information
 3  ON   'timing.log'      Timing information for 3D Meshing routines.
 4  OFF  terminal          Testing of video generation - if on then
                           video specific drawing is enabled and related debug
                           statements will be printed.
... (several lines deleted) ...
45  OFF  terminal          Pillow Sheet debugging
46  OFF  terminal          Paver breakout detection (expensive)

echo      = On
info      = On
journal   = On
warning   = On
logging   = On, log file = 'test.log'
```

Graphical Display Information

The list view command provides information about the current settings of various graphics parameters. Sample output from this command is shown in Table 3-18. See the description of the See “Graphics” on page 44. for a description of this information.

Memory Usage Information

Information about CUBIT’s memory usage can be obtained from the **list memory** command. An optional identifier can be specified which will restrict the output to the memory usage for those types of objects only. The command syntax is:

```
List Memory ['<object type>']
```

Sample output from the list memory command is shown in Table 3-19. This output is typically only of interest to CUBIT developers, so no interpretation of the output will be given here. If you need more details on this command, please contact one of the CUBIT developers..

Table 3-18 Sample Output from 'List View' Command

```
CUBIT> list view

...Current View Parameters
From: < 0.000000i 0.000000j 5.654066k> (absolute)
At: < 0.000000i 0.000000j 0.000000k>
Up: < 0.000000i 1.000000j 0.000000k>
View: < 0.000000i 0.000000j 1.000000k> (From - At, normalized)

Distance from 'from' to 'at' is 5.654066.
Displayed view size is 4.115823 horizontal by 4.115823 vertical.

Perspective Angle is 40.000000 degrees.
Drawing Mode is 'wireframe'.
AutoCenter ON, AutoClear ON, Axis OFF, Border ON.
```

Table 3-19 Sample Output from 'List Memory' Command

```
CUBIT> list memory

Dynamic Memory Allocation per Object
... (several lines deleted) ...
Object Name: DLLList

Object Size: 48      Allocation Increment: 4096

Allocated Objects: 4096 (bytes) 196608 (4% of Total)
Free Objects: 142 (bytes) 6816 (3%)
Used Objects: 3954 (bytes) 189792 (96%)

Object Name: ArrayMemory

Object Size: 32      Allocation Increment: 8192

Allocated Objects: 16384 (bytes) 524288 (12% of Total)
Free Objects: 2508 (bytes) 80256 (15%)
Used Objects: 13876 (bytes) 444032 (84%)

Total Memory Allocation Information (bytes)

Allocated Memory: 4153344
Free Memory: 358160 (8%)
Used Memory: 3795184 (91%)

Total non-pool ArrayBasedContainer memory allocation = 123338 (bytes)
Maximum non-pool ArrayBasedContainer memory allocated = 132415 (bytes)
```

▼ Picking

A limited command-line capability exists to pick geometric entities in the CUBIT model. The user issues the **Pick** command, then clicks on the entity to be identified; CUBIT then reports

the id number and name of the picked entity in the command window. The following picking commands are available:

Pick {curve|surface|lump|volume|body [list]}

If the **[list]** option is used, information about that entity is listed in the command window as if the **List** command had been issued.

▼ Help Facility

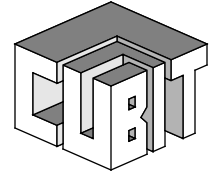


Two help systems are available in CUBIT: an window-based hypertext help facility and a text-based help facility. The user can access the hypertext help facility by typing **hyperhelp** and the text-based help facility by entering **help**. To attain further information about specific commands, the user can enter a specific keyword name after the term **help** or **hyperhelp**.

The text-based help facility prints only the usage syntax for the specified keyword. It is accessed by typing **help <keyword>**. For example, typing **help volume** would output the syntax for all volume commands to the standard output window. If no keyword is entered, a list of all valid keywords is output. The text-based help facility is accessed from the GUI version of CUBIT by typing help in the command line text field.

The window-based hypertext help facility is accessed by typing **hyperhelp <keyword>**. A window should pop-up containing a representation of the command index portion of this manual. Additional keyword parameters can be entered following the keyword to obtain more specific help. For example: **view at hyperhelp** would provide help on the **view at** command rather than general help on the **view** command.

The hypertext help facility can be accessed from the GUI version of CUBIT by selecting the **Help** menu item on the main menu or by clicking the **help** button on specific windows for help specific to those windows.



Chapter 4: Geometry

- ▼ Geometry Definition...67
- ▼ Geometry Creation...69
- ▼ Geometry Manipulation...78
- ▼ Geometry Decomposition...84
- ▼ Geometry Consolidation...85

This chapter describes methods available in CUBIT to produce and manipulate the geometry needed for meshing. Definitions of geometric entities and the structure of the nonmanifold geometry represented by CUBIT are given. This is followed by sections describing geometry importation, creation and modification. Geometry consolidation, the process of generating cellular topology from a manifold model, is also described.

▼ Geometry Definition

All geometric entities that exist in the CUBIT environment are represented by a solid model. In two dimensional modeling systems, a list of connected directional line segments is sufficient to provide a complete and unambiguous geometric definition. In three dimensions, only a solid model representation can guarantee complete and unambiguous geometry. All meshing tools available in CUBIT use this solid model geometry when generating the discretized representation of the geometry, i.e. the mesh. ACIS[®] is the solid modeling engine currently used by CUBIT. However, CUBIT uses its own geometric overlay to represent a non-manifold cellular topology for meshing.

Geometric Topology

Topology refers to the manner in which geometric entities are connected within the solid model. Within CUBIT, the geometric entities consist of *vertices*, *curves*, *surfaces*, *volumes*, and *bodies*. A collection of one or more of these entities can be created and is called a *group*. They are defined as follows:

Vertex

A vertex occupies a single point in space. A vertex is used to bound a curve and/or to specify a specific location for a node. A vertex which is located in the interior of a surface is called a hard point. It is used to force a node location to that specific geometric location.

Curve

A curve is a line (not necessarily straight) which is bounded by at least one but not more than two vertices. An example of a single vertex curve (both ends of the curve meet the same vertex) might be the curve which bounds an end cap surface of a cylinder. A curve is used to bound a surface. Curves may be generated independent of surfaces and as such used to specify the geometry for a sequence of bar or beam elements.

Surface

A surface in CUBIT is a finite bounded portion of some geometric surface (finite or infinite). A set of surfaces bound the volume in a volume. A surface is bounded by a set of curves. Surfaces must have at least one bounding curve to be meshed in CUBIT. Surfaces may be generated independent of volumes. Such free surfaces may be used to specify the geometry for shell elements. A periodic surface is a surface which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction.

Volume

Volumes are volumetric regions and are always bounded by one or more surfaces. For practical consideration, volumes will always be bounded by two or more surfaces. CUBIT currently cannot mesh a volume bounded by only one surface (e.g. a sphere) since such a surface has no bounding curves.

Body

A body is simply a collection or set of volumes. It differs from volumes only in the fact that booleans are only performed between bodies, not between volumes. The simplest body contains one volume.

Group

A group is a collection of one or more geometric entities (including other groups).

The command syntax to create or modify a group is:

group {id | "name"} add <list of geometry entities>

For example, the command

group "Exterior" add surface 1 to 2, curve 3 to 5

will create the group named **Exterior** consisting of the listed geometric entities. Any of the commands that can be applied to regular geometric entities can be applied to groups, for example, **mesh Exterior**, **list Exterior**, or **draw Exterior**. A geometric entity can be removed from a group using the command:

group <id> remove <list of geometry entities>

When a group is meshed, CUBIT will automatically perform an interval matching on all mapped, submapped, and trimapped surfaces in the group (including surfaces that are a part of volumes or bodies in the group).

Cellular Topology

Cellular topology (non-manifold topology) allows the connection of any number of surfaces to curves. A typical manifold model (or 2-manifold topology) only allows two surfaces to be bounded by a single curve. Cellular topology, because of its more general nature, allows two

adjacent volumes to share a common surface between them as shown in Figure 4-1. It also allows the formation of dangling faces and edges (Figure 4-2). These geometric constructs are sometimes useful in generating complex meshes.

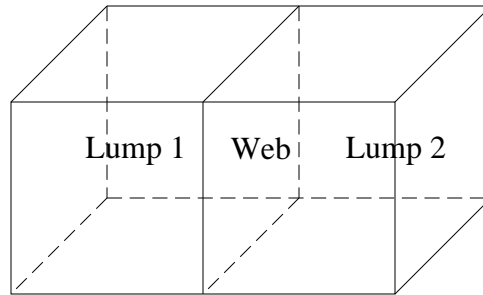


Figure 4-1 Cellular Topology Between Volumes

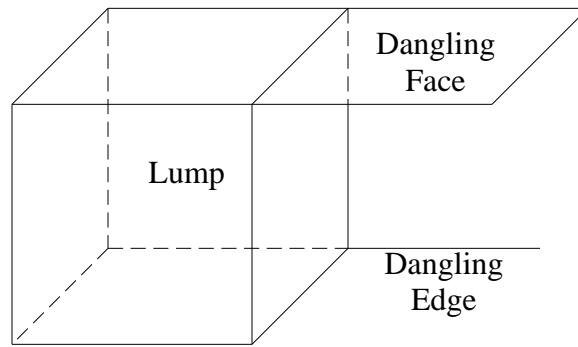


Figure 4-2 Dangling Faces & Edges

Cellular topology's advantage for mesh generation is that, when used properly, it eliminates the problem of equivalencing the mesh entities that are supposed to be shared between adjacent geometric entities (for instance, the common surface (Web) shown in Figure 4-1). It also supports the formation of dangling faces and dangling edges as shown in Figure 4-2. This allows the proper connection of surface, beam, and/or solid mesh elements.

▼ Geometry Creation

The examples in “Examples” on page 147 show solid model construction using several standard techniques. One technique is to use primitives (e.g. bricks, cylinders, spheres, etc.) and perform booleans (e.g. subtractions, intersections, and unions) on the primitives to generate the desired geometric shape. Another approach is to construct two-dimensional wireframe geometry and then perform a sweep (either along a path or about an axis) to create the three-dimensional solid model. As the examples indicate, it is necessary to save the solid model to a disk file (using a **.sat** filename extension) for subsequent meshing in CUBIT.

CUBIT geometry may be created within CUBIT using basic primitives and/or the sketchpad in combination with the boolean operators. The generation of geometric primitives and their manipulation and positioning will be discussed first in this section, with the boolean operations covered afterwards. Geometry may also be produced using several programs and imported into CUBIT through ACIS® format files. Several of these packages are discussed later in this section.

Since the geometry acts as the template for discretization into a mesh, care must be taken to insure an appropriate representation for the problem being analyzed. Users will find that building solid models of “real parts” will likely consist of a combination of the approaches mentioned above.

Geometry Primitives

Geometry primitives are classes of general geometric shapes which are differentiated by basic parameters. For example, a cylinder is a basic shape that can be specified by the parameters *height* and *radius*. Primitives available in CUBIT include the brick, cylinder, torus, prism, frustum, pyramid, and sphere. These primitives can be generated and used in boolean operations to produce very complex shapes. The geometric primitives can also be used in boolean operations with geometry generated through other means (e.g. using the sketch pad, or geometry read in from other sources). When using the GUI, the geometric primitives can be generated using the **Primitives** suboption in the **Geometry** menu. Figure 4-3 shows a sample of the available primitives.

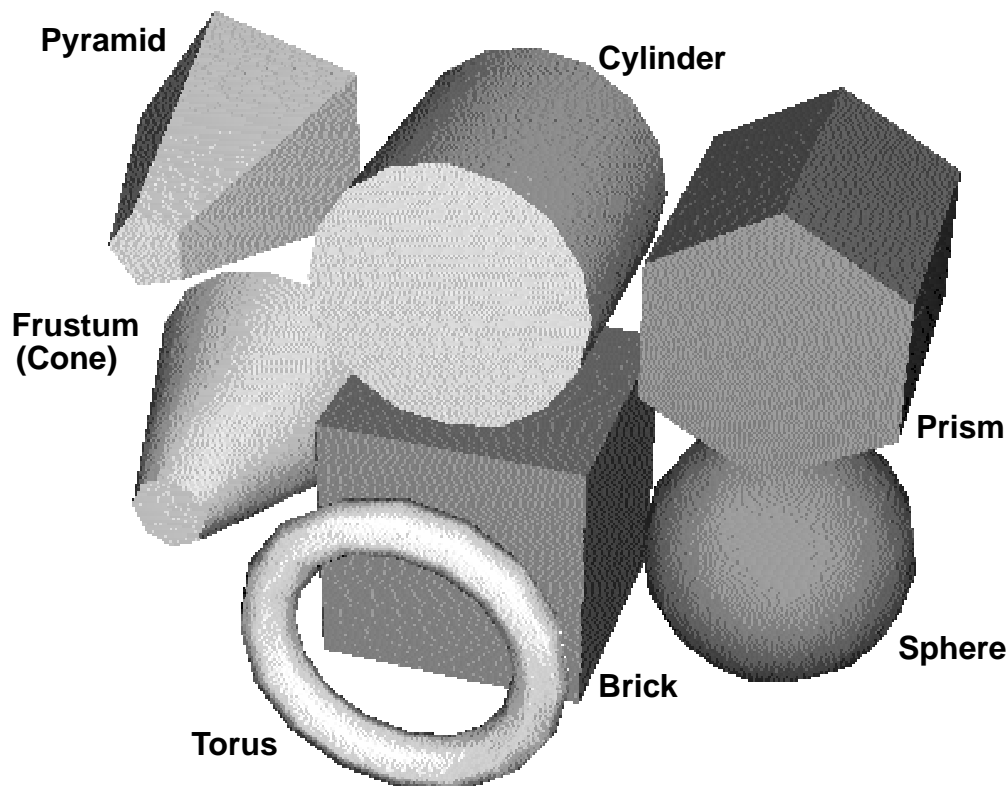


Figure 4-3 CUBIT Geometry Primitives

Brick

The brick is a cuboid where all the surfaces intersect at right angles. Figure 4-4 shows the **Geometry Primitives** dialog box when **Brick** is chosen. There are three parameters that may be specified, **Width** (x-dimension), **Depth** (y-dimension), and **Height** (z-dimension). Only **Width** is required. If only a **Width** value is specified, the other two values default to that value,

thus producing a cube. The **Apply** button will generate the brick. The equivalent command to generate a brick using the command line is

[create] brick width <x-dimension> [depth <y-dim> height <z-dim>]

[create] brick x <x-dimension> [y <y-dim> z <z-dim>]

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin and aligned with the coordinate axes.

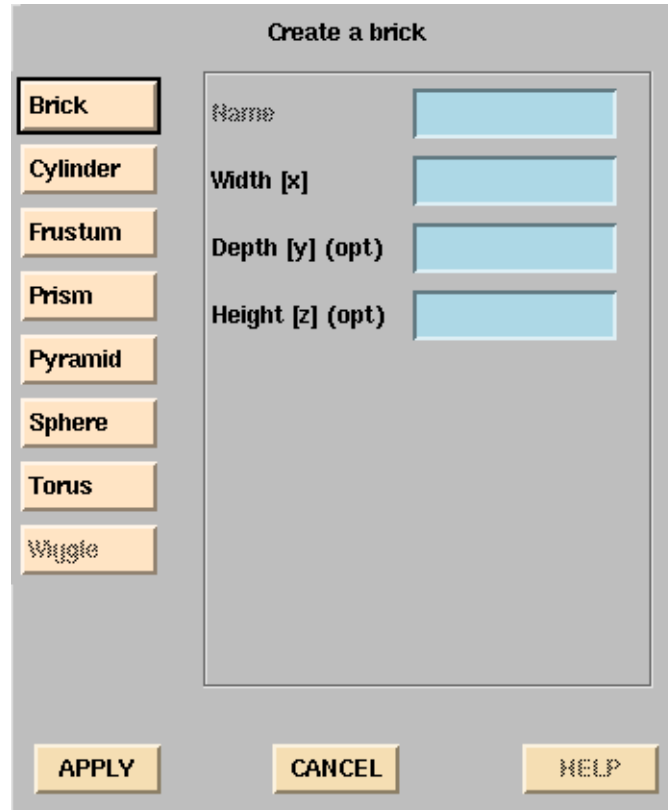


Figure 4-4 Brick Creation Dialog Box

Cylinder

The cylinder is a constant radius tube with right circular ends. Figure 4-5 shows the relevant portion of the **Geometry Primitives** dialog box when **Cylinder** is chosen. There are two parameters that must be specified, **Height** (z-dimension), and **Radius** (x/y-dimension). The **Apply** button will generate the cylinder. The equivalent command to generate a cylinder is

[create] cylinder height <z-height> radius <x/y-radius>

[create] cylinder z <z-height> radius <x/y-radius>

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin and aligned with the length of the cylinder along the z-axis.

Note: A cylinder may also be created using the **frustum** command with all radii set to the same value.



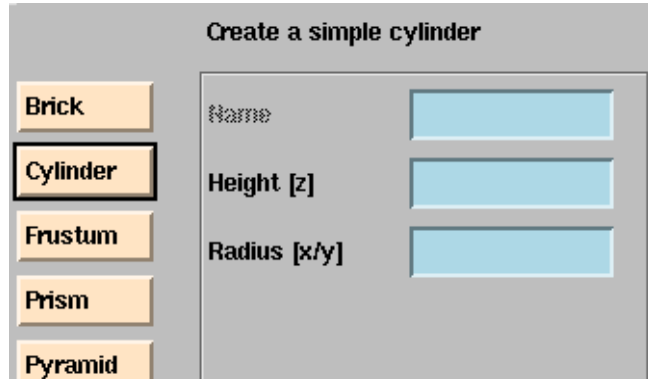


Figure 4-5 Cylinder Creation Dialog Box

Prism

The prism is an n-sided, constant radius tube with n-sided planar faces around the tube. Figure 4-5 shows the relevant portion of the **Geometry Primitives** dialog box when **Prism** is chosen. There are three parameters that must be specified, **Height** (z-dimension), **Sides** (number of sides) and **Radius** (x/y-dimension). The radius defines the circle circumscribing the prism cross-section. The **Apply** button will generate the prism. The equivalent commands to generate prisms are:

[create] prism height <z-height> sides <nsides> radius <x/y-radius>

[create] prism z <z-height> sides <nsides> radius <x/y-radius>

[create] prism height <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius>

[create] prism z <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius>

The new body contains one volume which will be given the next highest body ID number. The prism will be centered about the origin and aligned with the height of the cylinder along the z-axis. One of the planar sides will be perpendicular with the X-axis. The number of sides must be greater than or equal to three.

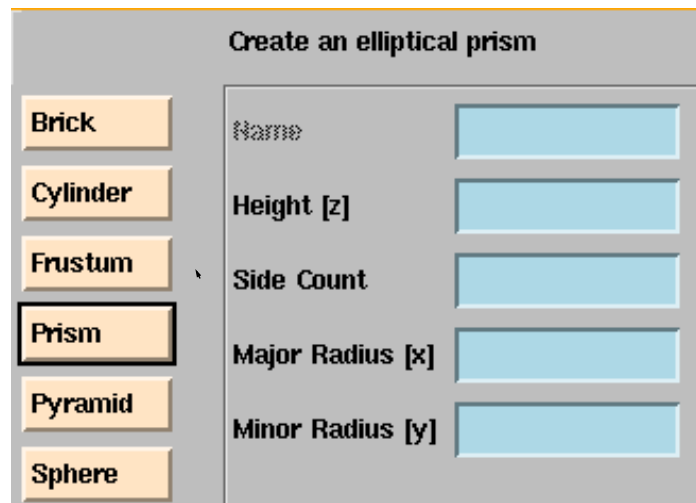


Figure 4-6 Prism Creation Dialog Box



Note: A prism may also be created using the **pyramid** command with all radii set to the same value.

Frustum

A frustum is a general elliptical right frustum. It can be thought of as a portion of a right elliptical cone. The elliptical nature comes by allowing a different radius in the two principle directions of the cone. Figure 4-7 shows the relevant portion of the **Geometry Primitives** dialog box when **frustum** is chosen. There are four parameters that may be specified, **Height** (z-dimension), **Major Radius** (x-radius), **Minor Radius** (y-radius) and **Top Radius** (x-radius at the top). The top y radius is calculated based on the ratio of the major and minor radii given. If only **Height** and **Major Radius** are specified, the other two radii are defaulted to the major radius value. If all radii are equal, a frustum defaults to a simple cylinder. The **Apply** button will generate the frustum. The command to generate a frustum using the command line is

```
[create] frustum height <z-height> major [radius] <x-radius>
minor [radius] <y-radius> top <top-x-radius>
```

```
[create] frustum z <z-height> major [radius] <x-radius>
minor [radius] <y-radius> top <top-x-radius>
```

The new body contains one volume which will be given the next highest body ID number. The frustum will be centered about the origin with the central frustum axis aligned with the z-axis.

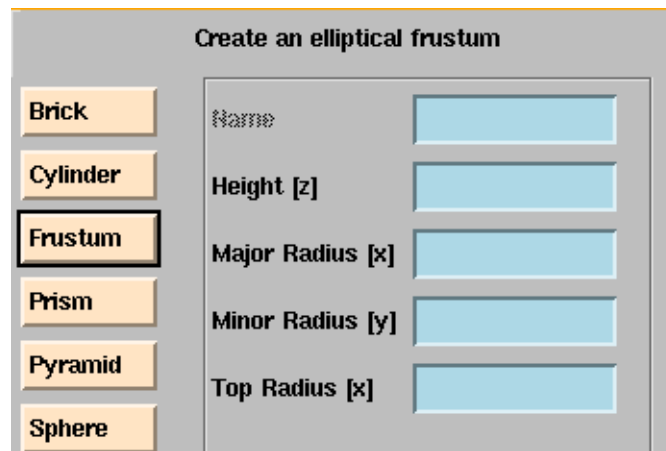


Figure 4-7 Frustum Creation Dialog Box

Pyramid

A pyramid is a general n-sided prism. It can be thought of as a portion of a right elliptical cone. The elliptical nature comes by allowing different circumscribing radii in the two principle directions of the pyramid. Figure 4-7 shows the relevant portion of the **Geometry Primitives** dialog box when **Pyramid** is chosen. There are five parameters that may be specified, **Height** (z-dimension), **Major Radius** (x-radius), **Minor Radius** (y-radius) and **Top Radius** (x-radius at the top). The top y radius is calculated based on the ratio of the major and minor radii given. If only **Height**, **Sides**, and **Major Radius** are specified, the other two radii default to the major radius value. If all radii are equal, a pyramid defaults to a simple n-sided prism. The

Apply button will generate the pyramid. The command to generate a pyramid using the command line is

```
[create] pyramid height <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius> top <top-x-radius>
```

```
[create] pyramid z <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius> top <top-x-radius>
```

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin with the central pyramid axis aligned with the z-axis.

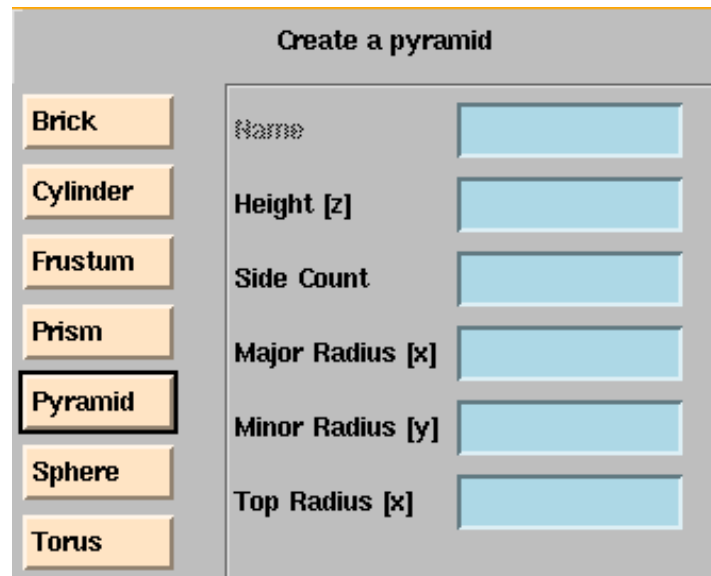


Figure 4-8 Pyramid Creation Dialog Box

Sphere

The **sphere** command generates a simple sphere. Figure 4-9 shows the relevant portion **Geometry Primitives** dialog box when **Sphere** is chosen. Only one parameter may be specified, **Radius**. The **Apply** button will generate the sphere. The command to generate a sphere from the command line is

```
[create] sphere radius <radius>
```

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin. Since portions of spheres are commonly generated, the capability to generate hemispheres, quadrants, and octants is provided. The syntax is

```
[create] sphere radius <radius> [inner radius] <inner_radius>
[delete] [xpositive] [ypositive] [zpositive]
```

If **inner_radius** is specified, a hollow sphere will be created with the specified inner radius. The identifiers **xpositive**, **ypositive**, and **zpositive** specify which portion of the sphere will be retained, or if the **delete** identifier is present, the portion of the sphere which will be removed. For example, to create an hemisphere in the positive x direction, enter the command:

```
sphere radius 5 xpositive
```

Torus

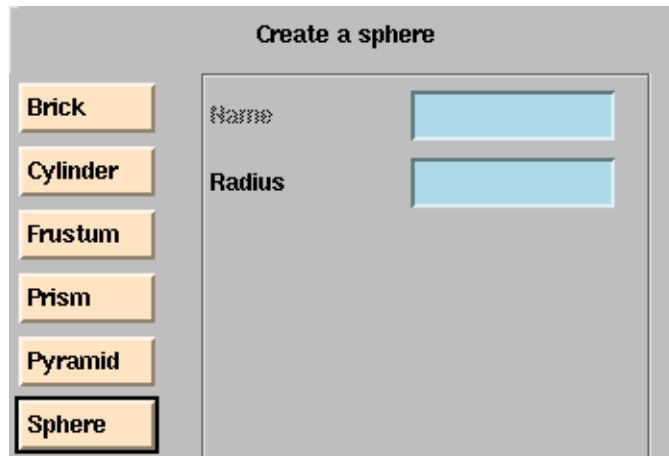


Figure 4-9 Sphere Creation Dialog Box

The torus command generates a simple torus. Figure 4-10 shows the relevant portion of the **Geometry Primitives** dialog when **Torus** is chosen. Two parameters must be specified, **Major Radius**, or the radius of the spine of the desired torus, and **Minor Radius**, or the radius of the cross-section of the ring. The minor radius must be less than the major radius. The **Apply** button will generate the torus. The command to generate a torus from the command line is

[create] torus major [radius] <major-radius> minor [radius] <minor-radius>

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin with the spline of the torus aligned perpendicular to the z-axis.

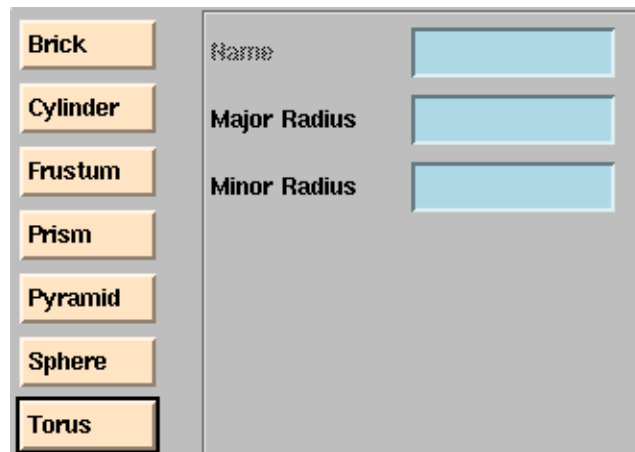


Figure 4-10 Torus Creation Dialog Box

Sketchpad Geometry



The sketchpad is used to construct two-dimensional wireframe geometry profiles. The profiles can be swept (either along a path or about an axis) to form more general solid bodies than geometry primitives. The swept bodies created with the sketchpad can be used in boolean operations with other geometry (e.g. primitives, other swept bodies, or geometry read in from other sources). The sketchpad is activated by selecting the **Sketch** suboption in the **Geometry** menu. The sketchpad is active when the sketch creation dialog box shown in Figure 4-11 is visible. The sketchpad is only available from the GUI version of CUBIT at this time.

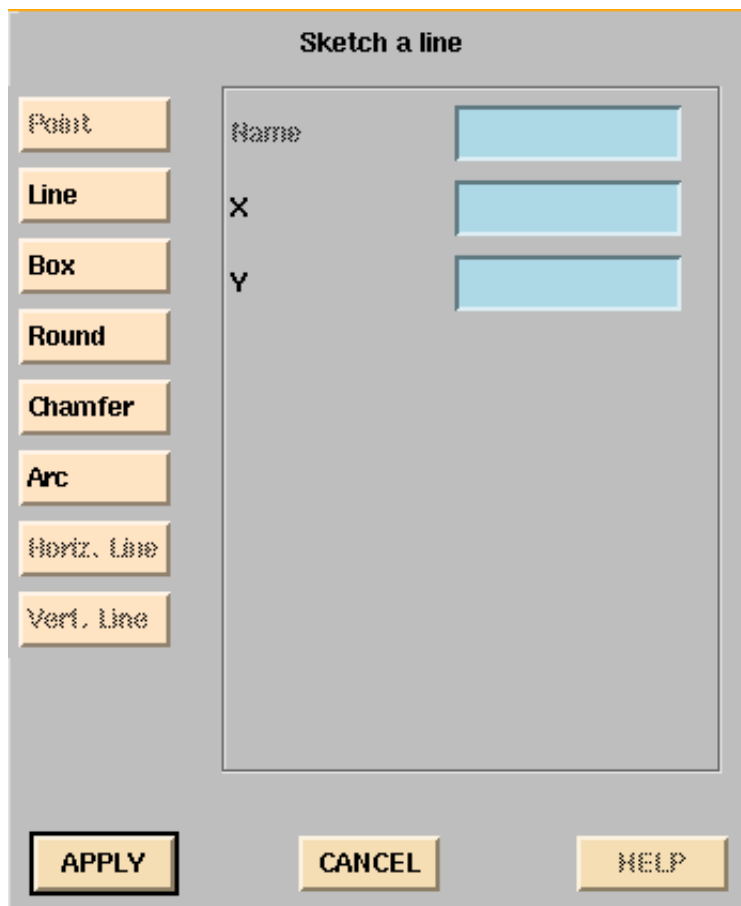


Figure 4-11 Sketch Creation Dialog Box

Sketch Overview

Creating a sketched profile is a two step operation: first define a polygonal outline and then refine the outline by chamfering or filleting corners or by pulling edges of the polygon into arc shape. The sketch primitives (line, box, round, chamfer, or arc) are created by clicking the corresponding button in the left column of the sketch creation dialog box. Then, in the graphics window, press the middle mouse button, drag the mouse to create the desired shape, and then release the mouse button. While dragging the mouse, the sketchpad provides rubber band feedback. When the sketched profile is complete, accept it by pressing the **Apply** button. The sketchpad will close the profile and create a two-dimensional wireframe geometry profile. The sketch can be aborted by pressing the **Cancel** button.

Polygonal Outline

The primary method for creating the polygonal outline is to use the **line** primitive. The first mouse press defines a starting point for the outline. Mouse release defines a straight **line**. If the sketched line is nearly horizontal or nearly vertical, the sketchpad will adjust the line. The other method for creating part of the polygonal outline is to use the **box** primitive. The box is a shortcut for sketching three lines in order to create a horizontal or vertical slot or tab. If the line created before the box is more closely horizontal than vertical, then the slot or tab created will be vertical. Similarly, if the box is attached to a roughly vertical face, the resulting slot or tab will be horizontal.

Outline Refinement

The polygonal outline can be refined by chamfering or rounding any corner. After selecting the appropriate option, **chamfer** or **round**, press the middle mouse button near the corner to be modified and drag the profile to see the effect of the modification. During drag, feedback is the same for chamfer and round. When the profile is finally created rounds will be correct.

Another way to refine the outline is to bend any line into an arc. After selecting the **arc** option, press the middle mouse button near the middle of a line to be modified and drag the profile to see the effect of the modification.

Importing Geometry

Capabilities exist to use geometry files generated using the ACIS[®] Test Harness, Pro/Engineer, or FASTQ. All geometry read into CUBIT must be in the form of ACIS[®] files. The ACIS[®] Test Harness can generate ACIS[®] files directly. Pro/Engineer and FASTQ files must first be translated into the ACIS[®] format before being usable with CUBIT.

Importing ACIS Files

Externally-generated ACIS[®] files can be read into CUBIT by selecting the **Import > Acis** options under the **File** menu in the GUI. The command line syntax for this command is:

Import Acis '<acis_filename>'

Note that the filename must be enclosed in single quotes.

ACIS Test Harness

The ACIS[®] test harness is a convenient method to generate geometry, since it is available on every platform which maintains an ACIS[®] license which includes all systems on which CUBIT is available. The test harness can be used to create geometric models via boolean operations (unions, differences, and intersections) and sweeps of wire profiles.

The ACIS test harness is a solid model construction program that supports English-like commands entered via the keyboard, and provides simple graphic output in a graphics window. Users should find it similar to FASTQ [5] and GEN3D [10] in that all commands entered during a session are saved in a monitor file (.mon extension) that can be subsequently replayed at a later time to re-generate the model. The graphics window is only displayed when a draw command is entered, thus simplifying batch processing (draw commands may be commented out with a '#' character). Additionally, Aprepro [13] may be used to evaluate any algebraic expression in the ACIS monitor file, allowing for parameterized geometry construction.

PRO/Engineer

The PRO/Engineer[®] product can also be used to create CUBIT geometry, although the support for this package is still being developed. Converters are being implemented which manage the translation of PRO/Engineer[®] assembly data into ACIS[®] format.

This solution is being pursued to address user needs and requirements regarding standard geometry formats at Sandia National Laboratories. Advantages to this creation method include the availability of the large number of parts being designed under the PRO/Engineer[®] format. PRO/Engineer[®] documentation and training is available at Sandia including consulting on an as-needed basis. Disadvantages include the single direction translation through which these parts must be sent to convert them to ACIS[®]. Some difficulties with numerical accuracy

involving PRO/Engineer® have been cited although no negative impact of this on CUBIT meshing functionality has yet been observed.

FASTQ



Support for reading a FASTQ file directly into CUBIT is available. FASTQ files are imported into CUBIT using the **Import Fastq** command or the **Import > Fastq** option under the File menu in the GUI. The command line command syntax is:

Import Fastq '<fastq_filename>'

All FASTQ commands are fully supported except for the **Body** command, some of the **Scheme** commands, and the non-circular arcs. At the current time, only the mapping, paving, and triangle primitive scheme commands are handled. The pentagon, semicircle, and transition primitives are not handled directly, but are meshed using the paving scheme. The FASTQ input file may have to be modified if the Scheme commands use any non-alphabetic characters such as '+', '(', or ')'. Circular lines with non-constant radius are generated as a logarithmic decrement spiral in FASTQ; in CUBIT they will be generated as an elliptical curve.

Since a FASTQ file by nature will be defined in a plane, it must be projected or swept to generate three dimensional geometry. Since this sweeping operation cannot currently be executed within CUBIT, only two-dimensional meshing of the FASTQ geometry can be performed. If three-dimensional geometry generated from the FASTQ geometry is required, the **fsqacs** translator should be used. It will generate ACIS files which can be used in the ACIS Test Harness and projected or swept to generate three-dimensional geometry which can be imported into CUBIT.

▼ Geometry Manipulation

Bodies can be translated, rotated, reflected, scaled, and copied in order to position them correctly before performing other tasks associated with generating a model. Boolean operations can also be performed between bodies. The transform operations which translate, reflect, scale, rotate and copy are described first in this section, followed by a description of the boolean operations intersect, subtract and unite.

Transform Operations

The translate, reflect, scale, and rotate functions do not create new geometry, whereas the copy, intersect, subtract and unite functions do create new geometry. If running the GUI, the **Transform** submenu of the **Geometry** menu is used for these operations.

Copy

The copy command copies an existing body to a new body without modifying the existing body. Figure 4-12 shows the dialog box that appears when the **Copy** option is chosen. Simply inserting a body ID in the **Body Name** and then pressing the **Apply** button will generate a new copy at the same location as the old one. When using the command interface, a copy can be

made of several bodies at once, and the group can be translated with a specified offset, or rotated about a given vector. The commands for copying a body from the command line are

body <range> copy [move <x-offset> <y-offset> <z-offset>]

body <range> copy [reflect {x | y | z}]

body <range> copy [reflect <x-comp> <y-comp> <z-comp>]

body <range> copy [rotate <angle> about {x | y | z}]

body <range> copy [rotate <angle> about <x-comp> <y-comp> <z-comp>]

body <range> copy [scale <scale-factor>]

If the **copy** command is used to generate new bodies, a copy of the original mesh generated in the original body can also be copied directly into the new body. This is currently limited to copies that do not interact with adjacent geometry. For details on mesh copies, see “Mesh Duplication” on page 114.

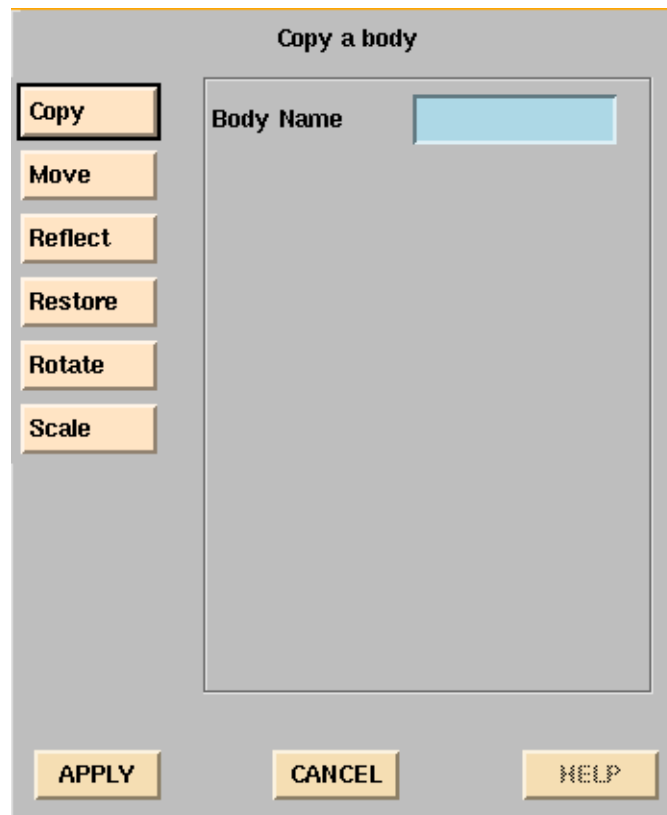


Figure 4-12 Body Copy Dialog Box

Move

The move command moves a body without adding any new geometry. Figure 4-13 shows the relevant portion of the dialog box that appears when the **Move** option is chosen. Inserting a body ID in the **Body Name** slot and filling in any of the offsets desired will move the body once the **Apply** button is pushed. Any **Offset** buttons not filled in are assumed to be zero. When using the command interface, a range of bodies to move can be specified

body <range> [copy] move <x-offset> <y-offset> <z-offset>.



Figure 4-13 Body Move Dialog Box

Scale

The scale command resizes the body without adding any new geometry. Figure 4-13 shows the relevant portion of the dialog box that appears when the **Scale** option is chosen. Inserting a body ID in the **Body Name** slot and filling in the desired scale will resize the body once the **Apply** button is pushed. The body will be scaled about its centroid. When using the command interface, a range of bodies to scale can be specified

body <range> [copy] scale <scale>

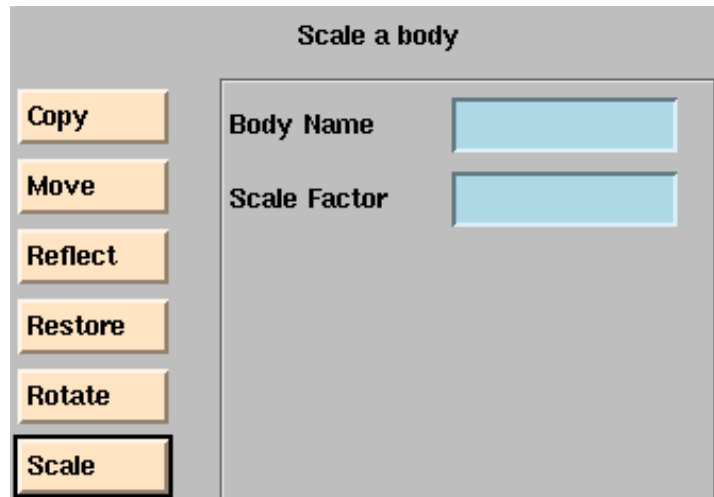


Figure 4-14 Body Scale Dialog Box

Rotate

The rotate command rotates a body about a given axis without adding any new geometry. Figure 4-15 shows the relevant portion of the dialog box that appears when the **Rotate** option is chosen. Inserting a body ID in the **Body Name** slot and filling in an **Angle** as well as any of the vector **Components** desired will rotate the body once the **Apply** button is pushed. If the **Angle** or any **Components** are not filled in they are defaulted to be zero. When using the

command interface, a range of bodies to be rotated can be specified, as well as a rotation about one of the cartesian axes:

body <range> [copy] rotate <angle> about {x | y | z}

body <range> [copy] rotate <angle> about <x-comp> <y-comp> <z-comp>

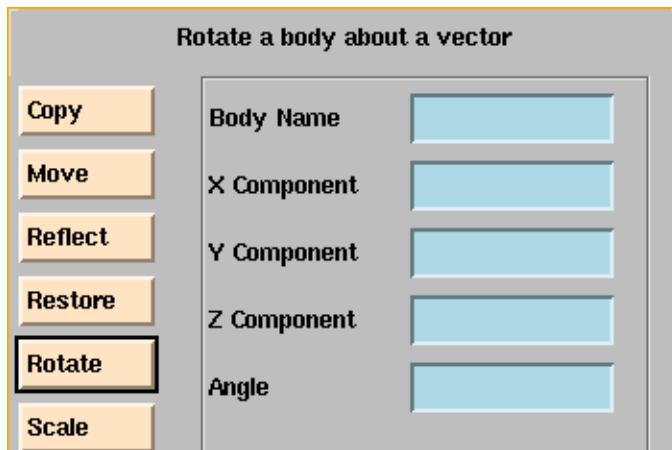


Figure 4-15 Body Rotate Dialog Box

Reflect

The reflect command mirrors a body about a plane normal to an arbitrary vector without adding any new geometry. Figure 4-13 shows the relevant portion of the dialog box that appears when the **Reflect** option is chosen. Inserting a body ID in the **Body Name** slot and filling in the components of the reflection vector reflects the body about a plane normal to this vector once the **Apply** button is pushed. Any vector **Component** buttons not filled in are assumed to be zero. When using the command interface, a range of bodies to be reflected can be specified as well as directly specifying a plane normal to one of the coordinate axes.

body <range> [copy] reflect <x-comp> <y-comp> <z-comp>

body <range> [copy] reflect {x | y | z}

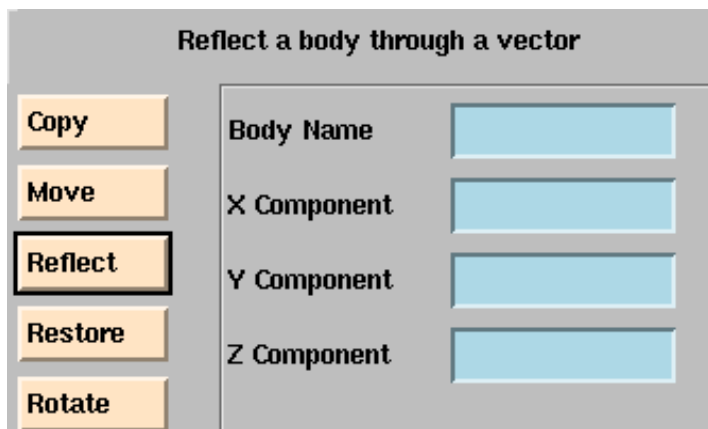


Figure 4-16 Body Reflect Dialog Box

Restore



The restore command removes all previous geometry transformations from the specified body. Execution of this command is similar to the copy command except that the Restore button is selected in the Dialog Box. When using the command interface, a range of bodies to be restored can be specified.

body <range> restore

Boolean Operations

Boolean operators allow boolean interactions (e.g. intersection, union, etc.) between bodies to produce a new body. The boolean operators available in CUBIT for modifying bodies are intersect, subtract and unite. When using the GUI, the boolean operators are available under the **Geometry** menu's **Booleans** option.

Intersect

The intersect command generates a new body composed of the portions of the geometry that are shared by both. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. Figure 4-17 shows the **Geometry Booleans** dialog when the **Intersect** option is chosen. The command line command syntax is

Intersect <body_id> With <body_id>

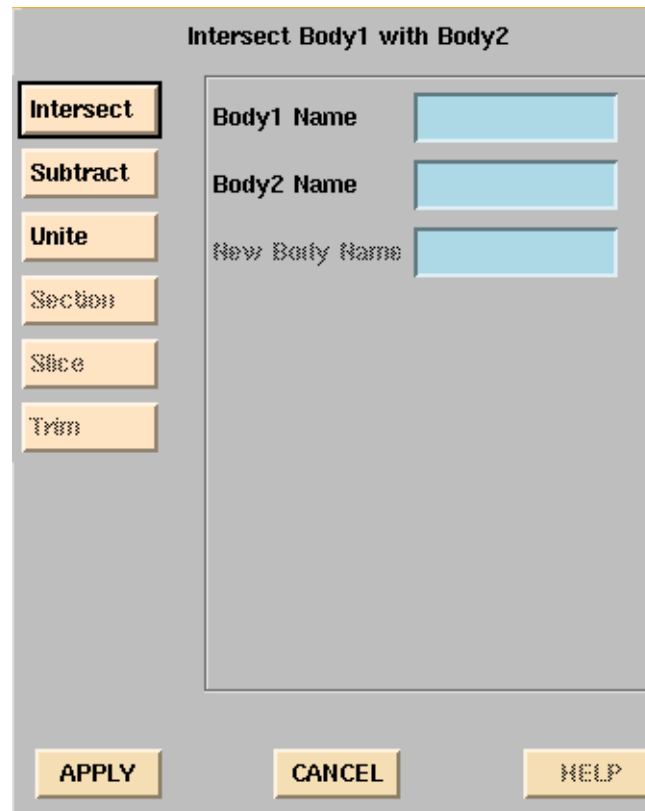


Figure 4-17 Intersect Boolean Dialog Box

Subtract

The subtract operator will subtract one body from another. The order of subtraction is significant. Both of the original bodies will be deleted and the new body will be given the next

highest body ID available. Figure 4-18 shows the relevant portion of the **Geometry Booleans** dialog box pertaining to subtract when the **Subtract** option is chosen. When the **Apply** button is pressed, Body2 is subtracted from Body1. The command line command syntax is

Subtract <body_id> From <body_id>.

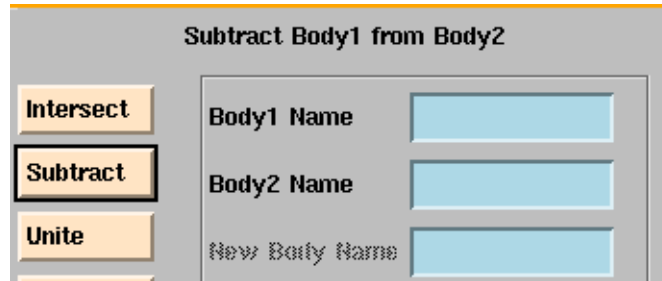


Figure 4-18 Subtract Boolean Dialog Box

Unite

The unite operator will combine two or more bodies into a single body. The original bodies will be deleted and the new body will be given the next highest body ID available. The command line command syntax is

Unite <body_id> With <body_id>

Unite Body <list_of_body_ids> [All]

The second form of the command permits the uniting of multiple bodies in a single operation. If the **All** identifier is entered instead of a list of body ids, all bodies in the model will be united into a single body.

▼ Geometry Decomposition

The capability to perform geometry decomposition within CUBIT is currently under active development. At the current time, geometry decomposition can only be performed crudely using boolean operations. A more robust and capable method called Web Cutting is not yet functional, but should be available in the near future.

Web Cutting



Web cutting allows the insertion of webs or faces between pieces of the geometry to facilitate meshing. Web cutting, although still being developed, should allow a user to decompose the geometry without the generation and use of boolean operations. This should greatly simplify the decomposition task.

When using the GUI, web cutting may be accomplished by choosing the **Decompose** option from the **Geometry** menu. A submenu will appear with the **Web Cutting** button. The dialog shown in Figure 4-19 will appear. A web is inserted by selecting a body, designating a plane,

The dialog box is titled "Web Cutting Dialog". It has a light gray background. At the top, there is a text box labeled "Body ID to Cut" with a selection arrow icon to its right. Below this is a section titled "Select A Plane Choice:" containing two radio buttons: "Face" and "3 Vertices". The "3 Vertices" button is selected and highlighted with a black border. Below the radio buttons are three text boxes labeled "Vertex 1 ID", "Vertex 2 ID", and "Vertex 3 ID", each with a selection arrow icon to its right. Below these is a section titled "Optional Vector:" containing two text boxes labeled "To Vertex" and "From Vertex". At the bottom of the dialog are three buttons: "APPLY", "HELP", and "CANCEL".

Figure 4-19 Web Cutting Dialog

and optionally specifying a vector. The body is supplied in the **Body ID to Cut** box (picking is optional). With the radio buttons in the area of **Select A Plane Choice** the desired mechanism for defining a plane can be designated. Selecting **Face** will require a face ID. Selecting **3 Vertices** will require 3 vertex IDs to define the plane. The **Optional Vertex** option allows the designation of a vector in the direction the cut is desired. Without this designation, the program will fire rays in multiple directions tangent to the plane to calculate an intersection loop. The ID of the vertex at the origin of the vector is entered in **From Vertex**. The ID of the vertex in the direction of the cut is entered in **To Vertex**. The **Help** button will

provide a brief help message if the help system is installed on the system. Web cutting is initiated by pressing the **Apply** button. The commands to perform web cutting using the command line are:

Webcut Body <body_id> Face <face_id> [Vector <from_vertex> <to_vertex>]

**Webcut Body <body_id> Vertex <vertex_1> Vertex <vertex_2>
Vertex <vertex_3> [Vector <from_vertex> <to_vertex>]**

The first command corresponds to using a face to specify the cutting plane and the second command uses three vertices to define the cutting plane. An optional vector defined by two vertices can be used to designate the direction of the cut.

Body-Based Decomposition

Primitive geometry decomposition can be performed using the geometry boolean operations available in CUBIT. An accelerated version of this type of operation is provided as well. If two ACIS bodies overlap in space, they can be decomposed into three separate bodies (one body containing the overlapping region and two bodies containing the non-overlapping pieces of the original bodies) using the **Decompose** command:

Decompose <body_id> With <body_id>

The three bodies resulting from this operation will have manifold surfaces; geometry consolidation should be used if these surfaces are to be represented with a single mesh (see “Geometry Consolidation” on page 85).

▼ Geometry Consolidation

Geometry consolidation is a means of enforcing mesh continuity and avoiding mesh node equivalencing. When a (manifold) solid model is constructed, it may contain any number of volumes which can belong to a single body. This does not imply that these volumes are “aware” of each other; CUBIT has no record or history of how the volumes from a body were created and what their spatial relationship may be. To determine this spatial relationship, some simple feature recognition concepts have been implemented to detect proximity between geometric entities, and to resolve any redundant entities.

Redundant geometry is not really redundant in a solid modeling sense, yet from a meshing standpoint (in which a contiguous mesh is required between volumes) if two surfaces exist where a single surface mesh must be generated, a redundancy must be resolved. This requirement can be illustrated in Figure 4-20. This figure depicts an “L” block (Figure 4-20), which could easily be meshed with a sweep operation, yet for our purposes here will be decomposed into two volumes (Figure 4-21) to demonstrate solid model surface redundancy.

For a contiguous mesh, CUBIT requires adjacent volumes to maintain one-to-one matching surfaces. The geometry consolidation tool in CUBIT is designed to recognize the spatial proximity between the surfaces and force them to use one surface mesh between them. This mesh sharing approach avoids the need to equivalence nodes and preserves mesh continuity. After the model in Figure 4-20 is decomposed into the model in Figure 4-21, surfaces 1 and 2 meet the one-to-one requirement. They can now be consolidated and treated as a single surface for meshing purposes. By extension, curves and vertices of adjacent volumes need to adhere to the one-to-one requirement also.

Assemblies which are imported from external solid modelers (e.g., Aries[®] ConceptStation, PRO/Engineer, ACIS[®] Test harness) will need to be inspected to insure that the adjacent volume

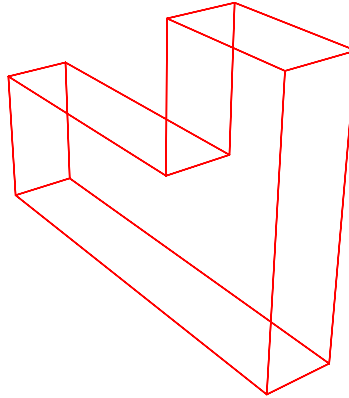


Figure 4-20 Solid Model Prior to Decomposition

surface to surface and curve to curve matching requirement has been properly adhered to. Some of these external modelers will use other internal geometry engines instead of ACIS[®] (PRO/Engineer does not run on ACIS[®]); when using geometry which was created in PRO/Engineer, it is critical to set the accuracy within PRO/Engineer to the highest level possible. The standard level of accuracy within ACIS[®] is much higher than that of PRO/Engineer (10^{-6} vs. 10^{-3}). CUBIT relies on geometric reasoning to identify matching surfaces and curves, and this technique is sensitive to the accuracy setting of the model being processed. Since Aries[®] ConceptStation is based on ACIS[®], it does not suffer from accuracy problems due to varying data formats and is probably the most attractive commercial modeler to use for creating geometry outside of CUBIT.

General Geometry Consolidation

When a model fully meets the one-to-one topology requirement, it is ready for full geometry consolidation of redundant geometric entities. The **Merge All** command searches first for matching surfaces, then for matching curves, and finally for matching vertices between adjacent volumes in the current CUBIT model. When a match is found, the consolidator merges the two

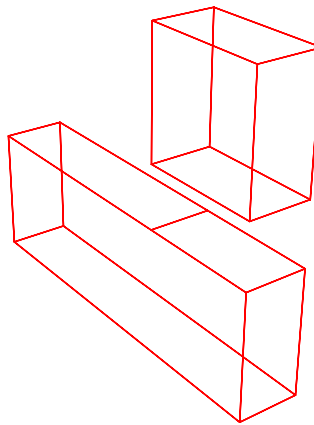


Figure 4-21 Solid Model After Decomposition

matching entities inside the CUBIT database and reconnects all the affected neighboring entities (such as now disconnected curves and surfaces) with the appropriate replacement entity. It is critical to complete geometry consolidation before any meshing of the affected geometry is initiated. If geometric entities are merged subsequent to meshing operations, some of the associated mesh entities may be lost. When using the GUI, selecting the **Geometry Consolidation** menu item from the **Special** menu will display the Geometry Consolidation Dialog shown in Figure 4-22. This dialog contains a **Geometry Consolidate** option menu which is used to describe the type of consolidation to be performed. If **All** is chosen from this option menu, general geometry consolidation is performed when the **Apply** button is pushed. General geometry consolidation is performed with the following command from the command line interface:

Merge All

Selective Geometry Consolidation

When analyses may require certain redundant geometric entities (such as slide lines requiring redundant curves), a more selective mode of geometry consolidation is available. Selective geometry consolidation can be initiated between any two user-specified surfaces, curves, or vertices. The consolidation process will be limited to the specified entities and their affected neighbor entities. The command syntax for selective geometry consolidation is as follows: **All Surfaces**, **All Curves**, **Surface**, and **Curve**. Below this are two text fields which are used to select specific geometric entities to be merged if **Surface** or **Curve** are selected in the option menu. The ID of the desired geometric entities can either be typed into the text fields, or the user can click on the arrows to the right of the text field to enable picking of the geometric entities. At this time, click the **Apply** button to execute the geometry consolidation process or click the **Cancel** button to exit from the dialog without executing any commands

Merge All Curves

Merge All Surfaces

Merge Curve <int_curve_id> **With** <int_curve_id>

Merge Surface <int_surface_id> **With** <int_surface_id>

Merge Vertex <int_vertex_id> **With** <int_vertex_id>.

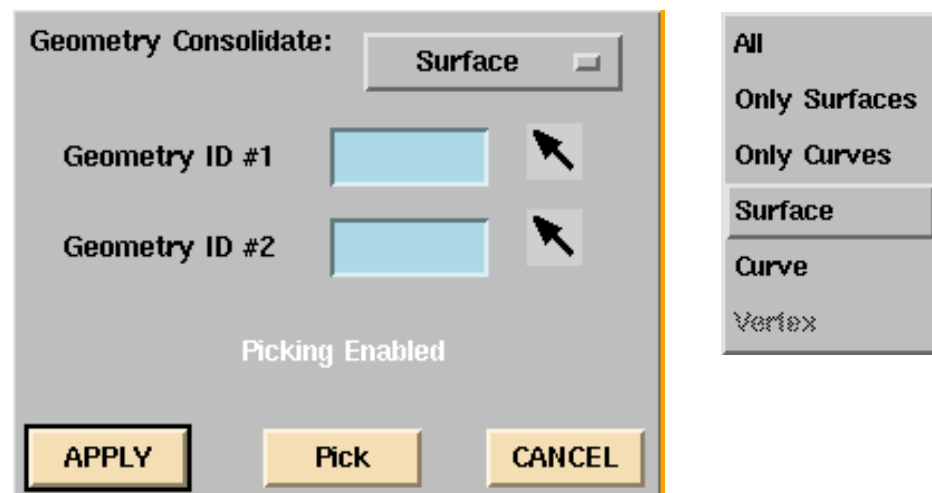


Figure 4-22 Geometry Consolidation Dialog Box

▼ Geometry Attributes

Each geometry entity has attributes which determine things like what color that entity is drawn in, and which meshing scheme shall be used to mesh that entity. This section describes geometry attributes which are not described elsewhere in this manual.

Entity Names

Geometry entities are assigned integer identification numbers in CUBIT in ascending order, starting with the number one. These numbers are not persistent; that is, each new entity created receives a unique id. However, geometry entities can be assigned names, and these names can be propagated explicitly by the user. The following commands assign names to geometry entities in CUBIT:

```
{Group | Body | Volume | Lump | Surface | Curve | Vertex}
Name '<entity_name>'
```

Geometry entities are given default names when they are first created. The default names consist of the names for the corresponding geometry entities (body, volume, surface, curve, or vertex), followed by the id number for that entity. So, for example, curve number 21 would have a default name of 'curve21'. The name for each geometry entity appears in the output of the **List** command.

Geometry entities can be identified either by the entity type followed by an identification number or by a unique name. A geometry entity name can be used anywhere that a entity type and id may be used. For example, if surface 3 is named CHAFER, the command 'mesh CHAFER' has the same result as the command 'mesh surface 3'.

Geometry entities may have multiple names, but a name may only refer to a single entity. If the geometry is read from an ACIS sat file which has the attributes **attrib-gtc-name**¹, the name of the owning entity will be set to that name. In a merge operation, the names of the deleted entity will be appended to the names of the surviving entity. The commands

```
label {body|volume|surface|curve|vertex} {name|id|on|off}
label geometry {name|id|on|off}
label {name|id|on|off}
```

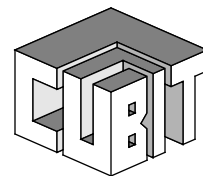
Control the type of labels displayed for an entity. If **name** is specified, the entities name will be used in the display; if **id** or **on** are specified, the entities id number will be displayed. The second and third forms of the command specify the labeling format for all geometry entities with a single command. The name of an entity can be set using the command

```
name {group|body|volume|surface|curve|vertex} <id> `entity_name'
```

A list of all names currently assigned and their corresponding entity type and id (optionally filtered by entity type) can be obtained with the command

```
list names [{group|body|volume|surface|curve|vertex|all}]
```

¹ The attribute used to specify the names in the ACIS sat file will probably change in the near future.



Chapter 5: Mesh Generation

- ▼ Mesh Definition...89
- ▼ Mesh Attributes...90
- ▼ Curve Meshing...95
- ▼ Surface Meshing...98
- ▼ Volume Meshing...107
- ▼ Mesh Duplication...114
- ▼ Mesh Editing...115

The methods used to generate a mesh using existing geometry are discussed in this chapter. The definitions used to describe the process are first presented, followed by descriptions of curve, surface, and volume meshing techniques. The chapter concludes with a description of the mesh editing capabilities.

▼ Mesh Definition

The mesh consists of entities similar in hierarchy to the geometry described in the previous chapter. The mesh entities include *nodes* (locations in space), *edges* (bar elements), *faces* (quadrilateral or shell elements), and *bricks* (hexahedral elements). Each mesh entity is associated with a geometry entity which owns it. This associativity allows the user to mesh, display, color, and attach attributes to the mesh through the geometry. For example, setting a mesh attribute on a surface affects all mesh entities owned by that surface.

Mesh Hierarchy

Mesh hierarchy refers to the manner in which mesh entities are connected within the mesh. Once a mesh is formed, the mesh entities define a discretized version of the geometry. The nodes required by higher-order elements are generated subsequent to the initial discretization; however, they are generated in the correct position based on the underlying geometry.

Node

A node is a single point in space. A node at a vertex is owned by that geometric vertex, a node on a curve is owned by that curve, a node on the interior of a surface is owned by that surface, and a node in the interior of a volume is owned by that volume.

Edge

An edge is defined by a minimum of two nodes. Additional nodes may exist on the edges of higher-order elements. An edge on a curve is owned by that curve, an edge in the interior of a surface is owned by that surface, and an edge in the interior of a volume is owned by that volume.

Face

A face is defined by four connected edges. A face on a surface is owned by that surface, a face in the interior of a volume is owned by that volume.

Brick

A brick is a hexahedral element defined by six connected faces. A brick is owned by the enclosing volume.

Mesh Generation

The mesh for any given geometry is generated hierarchically. For example, if the user issues a command to mesh a volume, first any unmeshed vertices are meshed with nodes. Next, any unmeshed curves are meshed, followed by the meshing of unmeshed surfaces. Finally the mesh within the volume is generated. Vertex meshing is of course trivial and thus the user is given no control over this process. However, curve, surface, and volume meshing can be directly controlled by the user. The **scheme** command specifies the meshing algorithm which will be used in meshing each of these geometric entities and the block command specifies the type of elements which will be generated by the meshing algorithm.

▼ **Mesh Attributes**

Each geometric entity has mesh attributes which specify information such as meshing scheme, discretization density, discretization distribution (equal or biased), and element type. Unless otherwise specified by the user, the default meshing attributes listed in Table 5-1 will be used.:

Table 5-1Default Meshing Attributes

<i>Geometric Entity</i>	<i>Default Attributes</i>		
	<i>Scheme</i>	<i>Element Type</i>	<i>Intervals</i>
Curve	Equal	2-node Bar	1
Surface	Map	4-node Quadrilateral	—
Volume	Map	8-node Hexahedron	—

Meshing Schemes

The mesh scheme attribute specifies the meshing algorithm that will be used to generate the mesh on each geometric entity. Note that the meshing scheme can be specified independently for curves, surfaces, and volumes; however, the meshing schemes for all surfaces on a volume must be compatible with the meshing scheme specified for that volume. For example, the Project, Translate, and Rotate volume meshing schemes require that some of the volume

surfaces be meshed using the Mapping scheme. The currently supported meshing schemes are listed in Table 5-2.

Table 5-2 Valid Meshing Schemes for Curves, Surfaces, and Volumes

<i>Scheme</i>	<i>Description</i>
Curve Meshing Schemes	
Equal	linear distribution of nodes along a curve based on arc length of the curve (default)
Biased	the distribution of nodes along a curve by biasing the nodal positions toward one of the curve ends given a growth factor
Surface Meshing Schemes	
Map	standard surface mapping transformation [7] (default)
SubMap	pseudo geometry decomposition of surfaces to produce block characteristic meshes
TriMap	generate triangular elements at sharp corners or specified vertices and mesh the remaining surface using the standard mapping transformations
Pave	advancing front algorithm for general surfaces including those with holes [1]
TriPave	generate triangular elements at sharp corners or specified vertices and mesh the remaining surface using the paving algorithm
Triangle	meshing primitive for three-sided regions
Volume Meshing Schemes	
Map	standard volumetric mapping transformations [7] (default)
Project	2&1/2D Sweeping Algorithm—general purpose
Translate	2&1/2D Sweeping Algorithm—along a vector
Rotate	2&1/2D Sweeping Algorithm—about a central axis
Plaster	Research algorithm for automatic hexahedral volume meshing
Weave	Research algorithm for automatic hexahedral volume meshing

Interval Specification

Interval settings control the discretization density of the generated mesh. The number of intervals, or discretizations, can be set on a body, volume, surface, or curve. A related setting, the interval size, specifies the *length* of element edges on a curve, rather than the number of intervals. An advantage of the interval size option is that consistent element sizes can be specified throughout the mesh.

Element Types

CUBIT supports several element types, including bars, beams, quadrilaterals, shells, and hexahedrons. Two- and three-node bar and beam elements; four-, eight-, and nine-node quadrilateral and shell elements; and eight-, twenty-, and twenty-seven node hexahedral elements are available. Multiple element types can be used in a single CUBIT model. The **Block**

commands are used to set element types. These are described in “Element Block Specification” on page 124. The local element node numbering is as specified in the Exodus II specification and shown in Figure 5-1..

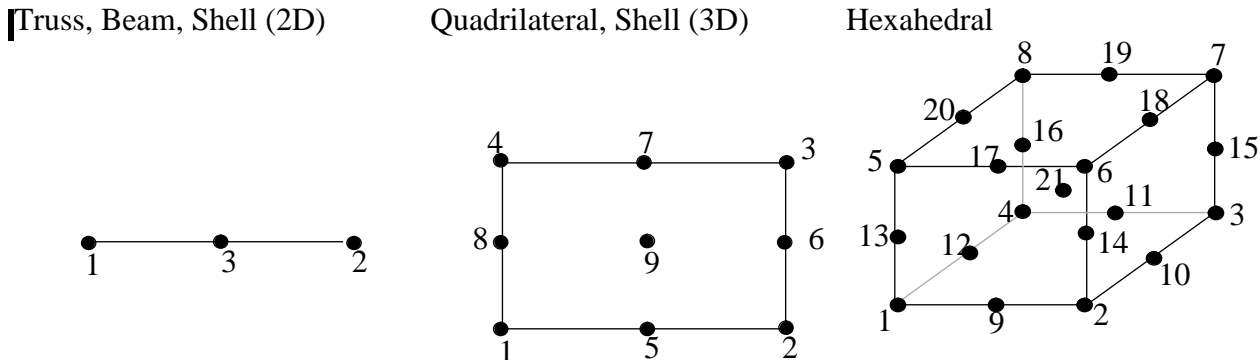


Figure 5-1 Local Node Numbering for CUBIT Element Types



Note: The type of elements to be generated on a geometric entity must be specified prior to meshing the geometric entity unless the defaults listed in Table 5-1 are desired.

▼ Surface Vertex Types

Often meshing algorithms, in particular algorithms based on the mapping process, must classify the vertices of a surface or volume to produce a high quality mesh. For example, a surface mapping algorithm must identify the four vertices of the surface that best represent the surface as a rectangle. The submapping, triangle primitive, trimap, and tripave meshing schemes have similar vertex identification needs. Although the surface vertex type can usually be assigned automatically, there are sometimes ambiguous cases or special cases in which the user needs to manually specify the classification of a particular vertex. The command

Surface <surface_id> Vertex <vertex_id> Type {end|side|corner|reversal}

Surface <surface_id> Vertex <vertex_id> Type {triangle|notriangle}

is used to manually specify the classification of a particular surface vertex. Note that a vertex may be connected to several surfaces and its classification can be different for each of those surfaces. Figure 5-2 illustrates the vertex angle types. Note that one element will be inserted at an end vertex, two elements at a side vertex, three elements at a corner vertex, and 4 elements at a reversal vertex.

Note: The Surface Vertex Type command does not need to be given in order to mesh a surface, however in some cases, it can improve the quality of the generated mesh.

▼ Automated Interval Assignment

Appropriate interval assignment is critical to produce high quality meshes using the map, submap, and triangle meshing schemes. When the scheme designation for a surface is *Map*, *Submap*, or *Triangle*, or the scheme designation for a volume is *Map*, *Submap*, *Project*,

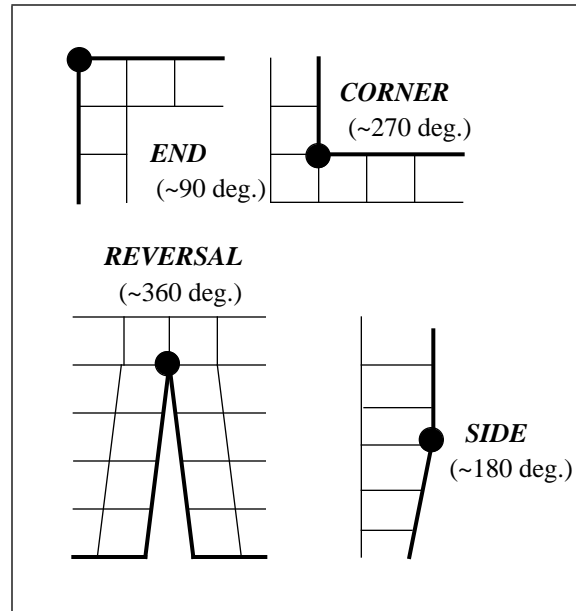


Figure 5-2 Illustration of Angle Types

Translate, or *Rotate* automated interval assignment tasks are performed prior to meshing the surface or volume. To perform automated interval assignment tasks on a group of surfaces or on a group of volumes the following commands are appropriate:

surface <range> match intervals

surface <list of surfaces> match intervals

volume <range> match intervals

The *Map*, *Submap*, *Project*, *Translate*, or *Rotate* meshing algorithms automatically execute the automated interval assignment algorithm and do not require explicitly issuing the match intervals command unless it is desired to match intervals on a group of surfaces or group of volumes simultaneously. Both the volume meshing commands and the manual (match intervals) command formulate a list of surfaces that are to have automated interval assignment tasks performed (based on meshing scheme). The list of surfaces is then sent to the automated tool which determines dependencies between intervals on curves and assigns compatible intervals according to the meshing scheme.

The automated interval assignment algorithm calculates a solution as close as possible to the initial specified intervals settings, while satisfying dependencies and compatibility constraints. When the actual *number* of intervals is specifically set (curve by curve) by the user, the intervals are designated as “hard set” and are not adjusted. When the interval *size* is specified, it is assumed that the user is specifying an approximate number of intervals to be placed on a curve rather than an absolute number. These are designated as “soft set” intervals and are taken as a lower bound by the automated interval assignment algorithm. If the number of intervals on a curve is not specified by the user it defaults to a “soft set,” one interval. Adjustments to “soft set” curves are minimized by the automated interval assignment algorithm while satisfying dependencies and compatibility constraints.

The automated interval assignment algorithm is designed to find one feasible solution among the possibly infinite number of possible interval solutions. To improve the chances of desired results, the user can specify a desired size for surfaces. This allows the automated interval

assignment algorithm a larger degree of freedom to make needed adjustments while having an initial desired target value to work from.

The following sections define the constraints that the automated interval assignment algorithm follows for the Map, Submap, and Triangle meshing schemes.

Scheme Map Interval Assignment Constraints

For surfaces with mapping schemes, first the designation of a “*logical rectangle*” fit to the surface is made. The mapping algorithm selects four vertices that will best transform the surface into a logical quadrilateral. These four vertices are chosen as “*logical corners*” and curves falling between these vertices are grouped as a “*logical side*.” In Figure 5-3, the *logical corners* selected by the algorithm are indicated by arrows. Between these vertices the *logical sides* are defined (see Table 5-1).

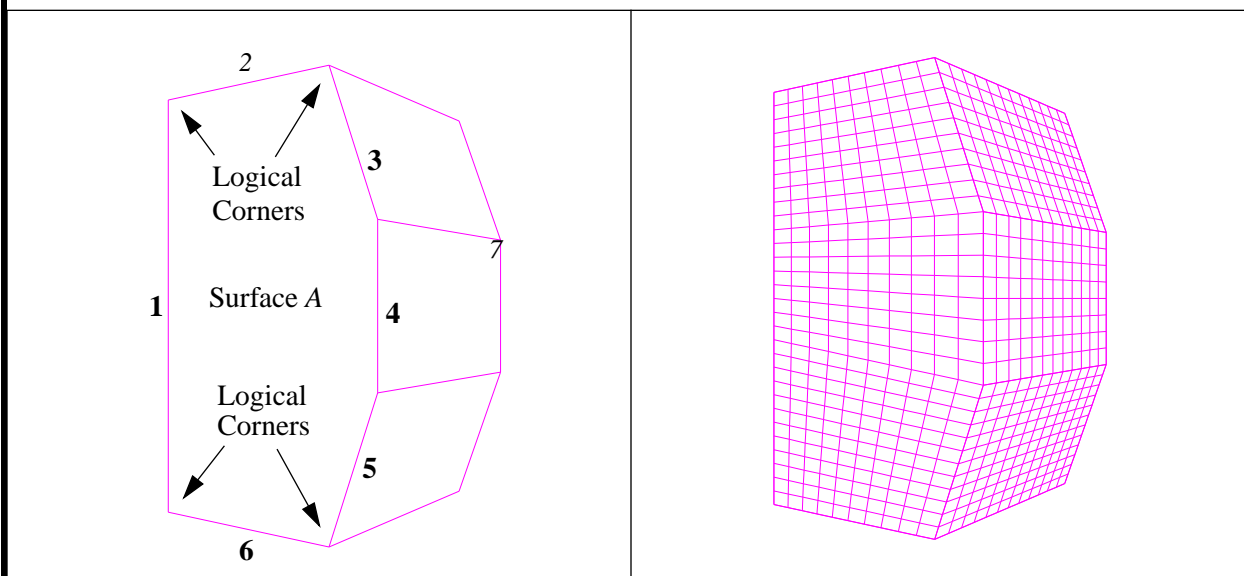


Figure 5-3 Scheme *Map* Logical Properties

Table 5-1Listing of logical sides

Logical Side	Curve Groups
Side 1	Curve 1
Side 2	Curve 2
Side 3	Curve 3, curve 4, curve 5
Side 4	Curve 6

Interval divisions on opposite sides of the logical rectangle are matched to produce the mesh shown in the right portion of Figure 5-3 (i.e. The number of intervals on logical side 1 is equated to the number of intervals on logical side 3).

The process is similar for volume mapping except that a logical hexahedron is formed from eight vertices.

Scheme Submap Interval Assignment Constraints

For surfaces with scheme submap, the designation of the “*logical rectangle*” is different from

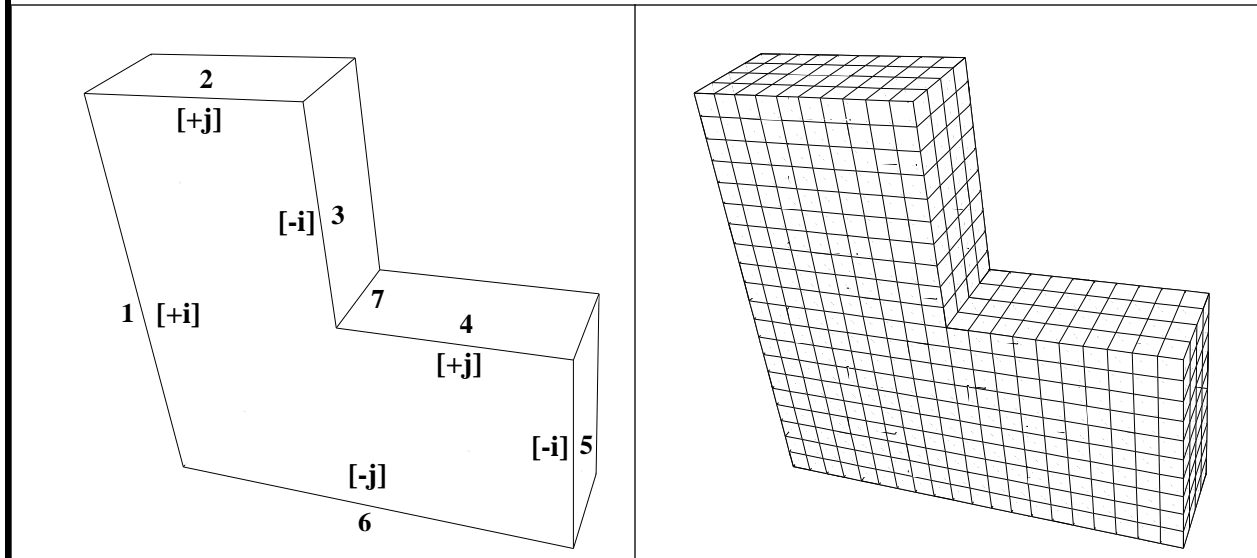


Figure 5-4 Scheme Submap Logical Properties

that of scheme map rectangle. Curves on the surface are traversed and grouped into “*logical sides*” by a classification of the curves position in a local “*i-j*” coordinate system. The local “*i-j*” coordinate system is defined by a traversal of the surface boundary curves. The traversal of the boundary and classification of curves is based on the interior surface angles at each vertex on the surface. Example, curve 1 may be arbitrarily defined in the coordinate system as [+i], a 90 degree turn defines curve 2 as [+j], another 90 degree turn defines curve 3 as [-i], a 270 degree turn defines curve 4 as [+j] and so on. The logical sides are then defined by grouping all curves with the same classification into one side. Therefore all [+i’s] are grouped as one side and all [+j’s] are grouped as another side and so forth. These four sides then define the “*logical rectangle*” that is used to formulate constraint equations (i.e. side 1 [+i’s] are equated to side 3 [-i’s] and side 2 [+j’s] are equated to side 4 [-j’s]).

▼ Curve Meshing

Curve meshing discretizes the curve, creating nodes and edges using the Mesh Dialog Box shown in Figure 5-5. For curve meshing, the **Geometry Type** must be set to **curve**. During curve meshing the user controls the density of nodes/edges (or intervals) along the curve and the relative spacing or bias of the nodes along the edge.

Node Density

The density of edges along curves is specified by setting the actual number of intervals or by specifying a desired average interval size. When the actual *number* of intervals is specifically set (curve by curve) by the user, these intervals are designated as being “hard set” and are never adjusted by the meshing algorithms—even when such an adjustment may produce a better mesh. It is assumed that when the interval *size* rather than an actual *number* of intervals is specified, the user is specifying an approximate number of edges to be placed on a curve rather

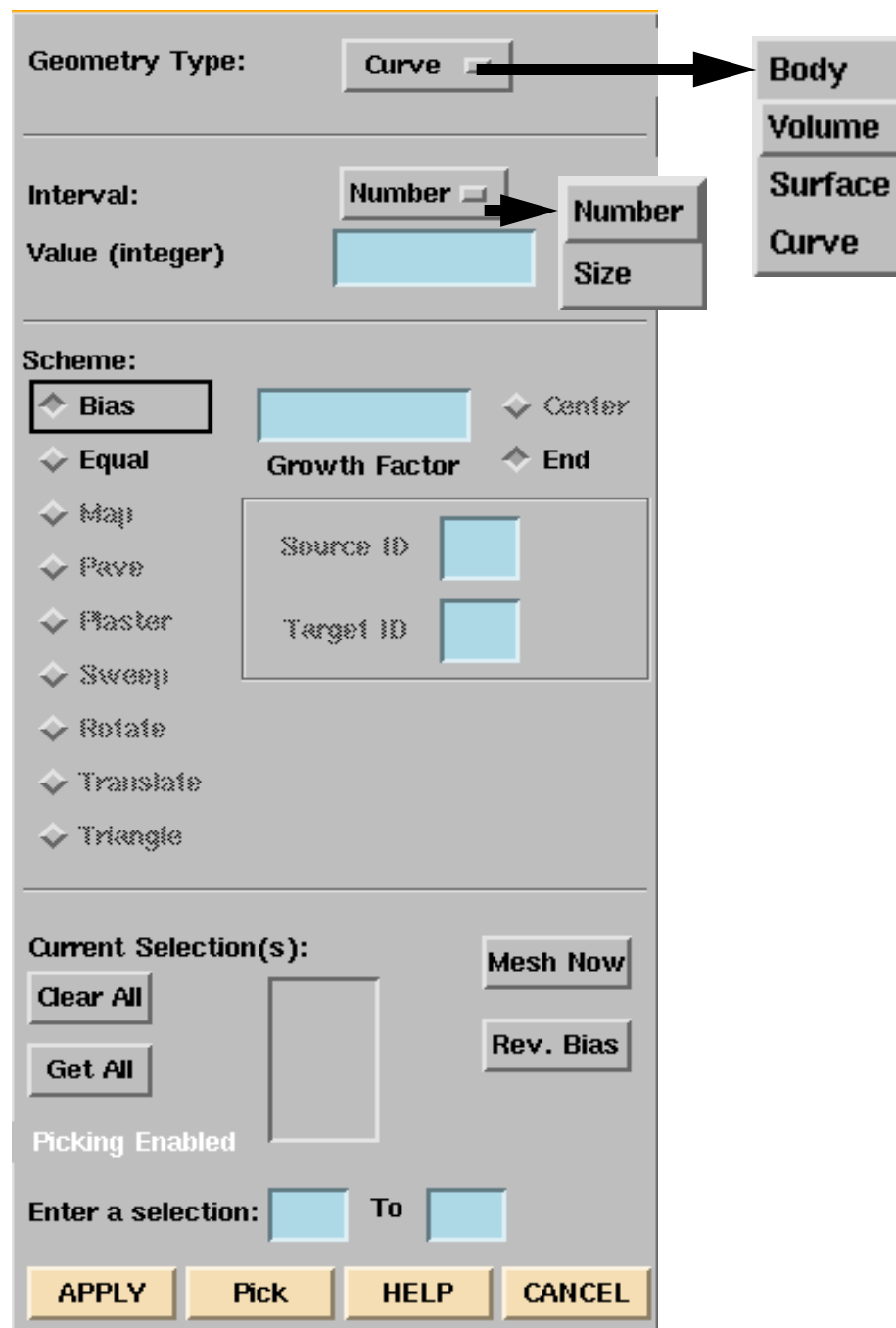


Figure 5-5 Curve Meshing With The GUI Mesh Dialog Box

than an absolute number. The meshing algorithms in CUBIT will then make minor adjustments to the number of intervals to insure an even number around the edge of a surface¹ and may also modify intervals so the chosen algorithm will have a higher probability of success. The number of intervals on a curve will not be adjusted after that curve has been meshed, either explicitly or

1. Meshing a surface or volume with an all-quadrilateral or all-hexahedral mesh requires an even number of intervals around the boundary of that surface or volume.

as the result of meshing a surface or volume containing that curve. Intervals can be changed if the existing mesh is first deleted. The number of intervals or interval size can be explicitly set curve by curve, or implicitly set by specifying the intervals or interval size on a surface or volume containing that edge. For example, setting the intervals for a volume sets the intervals on all curves in that volume. The default number of intervals is 1. Note that if higher-order elements are being generated, the number of intervals and interval size refer to the edge of an element, not necessarily to the spacing of nodes along that edge.

When using the GUI to specify the number of intervals, the **Interval** button must be set to **Number** and an integer supplied in the **Value** dialog box. *To actually apply this interval setting to the geometry, the **Apply** button must be pushed.* When setting the interval size, the **Interval** button must be set to **Size** and an appropriate size entered into the **Value** dialog box. *Again, to actually apply this interval size to the curve, the **Apply** button must be pushed.*

The equivalent commands to specify the number of intervals in the command line interface are:

{curve|surface|volume|body} <range> interval <intervals>

where **range** may be a single integer or a range of integers. Interval size may be specified in the command line interface using similar commands:

{curve|surface|volume|body} <range> size <interval_size>

Relative Element Edge Lengths

The relative length of element edges along a curve is specified using the curve scheme. Two curve schemes are currently supported: **equal** and **bias**. The **equal** scheme generates elements with equal length edges along the curve. The **bias** scheme requires the specification of a *bias factor* which designates a geometric progression of element edge lengths along the edge. For example, a bias factor of 0.9 will make (as much as possible) each edge along the curve 0.9 times the length of the previous edge, starting at the first vertex¹. The default bias factor is 1.0.

When using the GUI to specify the scheme, either the **Equal** or **Bias** radio button must be chosen. The curve scheme Map is equivalent to equal. If **Bias** is chosen, a bias factor is supplied in the **Growth Factor** dialog box. To actually apply this scheme setting to the geometry, the **Apply** button must be pushed. The command used to specify the curve scheme in the command line interface is:

curve <range> scheme {equal | bias} factor <factor>

The **factor** must be provided if using the bias scheme. To reverse the bias direction of a curve in the GUI, use the **Rev. Bias** (reverse bias) button on the curve meshing dialog. The command used to reverse the bias in the command line interface is:

curve <range> reversebias

Reversing the curve bias using this command is equivalent to setting a bias factor equal to the inverse of the original bias factor.

Sizing Function-Based Node Placement

The ability to specify the number and location of nodes based on a general field function is also available in CUBIT. With this capability the node locations along a curve can be determined by some field variable (e.g. an error measure). This provides a means of using CUBIT in adaptive

1. The first vertex of a curve can be determined with the list curve <id> command.

analyses. To use this capability, a sizing function must have been read in and associated to the geometry (see “Adaptive Surface Meshing” on page 101 for more information on this process). After a sizing function is made available, the following scheme will mesh the curve adaptively:

curve <range> scheme stride

Meshing the Curve

Once the appropriate interval and scheme settings have been made, the curve can be meshed. If using the GUI, the **Mesh Now** button is pushed to mesh the curve.



Note: *The **Apply** button must be pushed before the settings shown in the dialog are stored on the geometry and used when the **Mesh Now** button is pushed.*

If using the command interface, the appropriate command is:

mesh curve <range>

The resulting mesh will be drawn on the screen in the mesh color designated for the curve. Figure 5-6 shows the result of meshing two edges with equal and bias schemes.

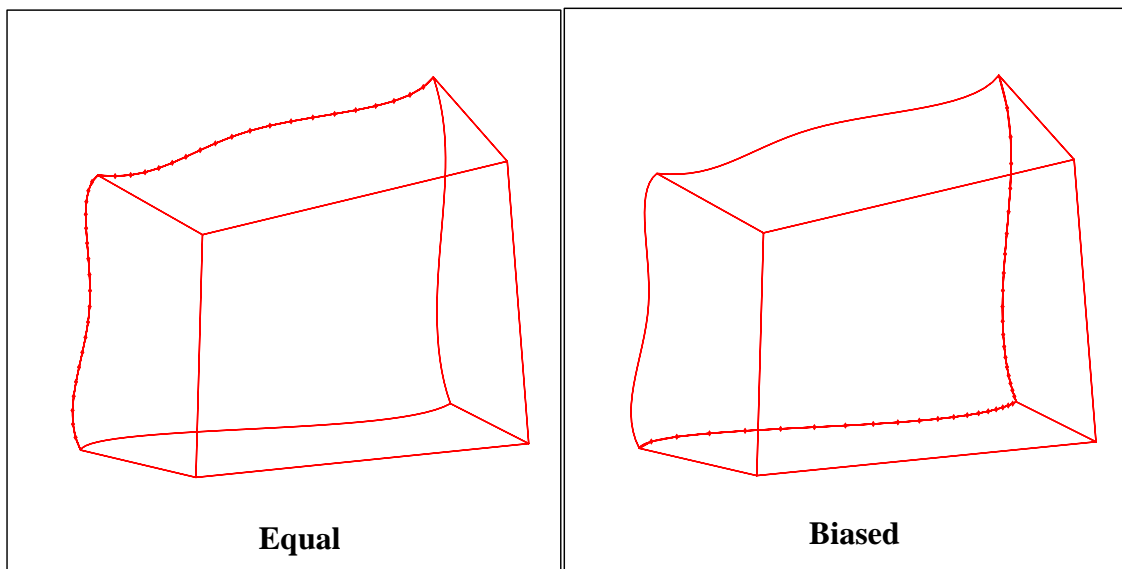


Figure 5-6 Equal and biased curve meshing

▼ Surface Meshing

Surface meshing discretizes a surface into nodes, edges, and faces. When meshing a surface, the bounding curves of the surface are first meshed (if not already meshed). The nodes on those curves are then used as the initial data for the surface meshing. Surface meshing algorithms include mapping, primitives, and paving, and a technique to apply boundary layers, or rows of aspect-controlled elements, to the surface before using the chosen algorithm.

When using the GUI, the portion of the **Mesh** dialog, from the **Generate** option shown in Figure 5-7 is used for surface meshing. The **Geometry Type** must be set to **Surface**. The

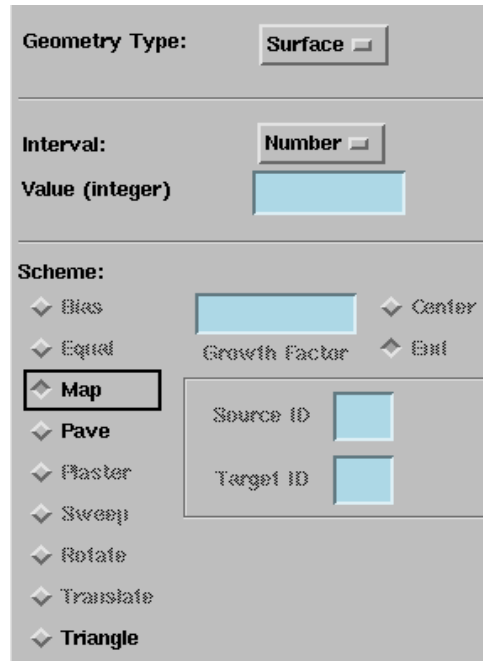


Figure 5-7 Surface Meshing with the GUI Mesh Dialog

Node density and relative spacing along the curves which bound the surface can be set as explained in “Curve Meshing” on page 95.

Scheme Designation

The algorithm to be used for surface meshing is designated as the scheme of the surface. Currently supported surface mesh schemes are **Map** (mapping algorithm), **Pave** (paving algorithm), **Submap** (mapping algorithm with geometry decomposition), **TriMap** (generate triangular elements, map remainder), **TriPave** (generate triangular elements, pave remainder) and **Triangle** (triangle primitive). Each of these algorithms are briefly described below. The default scheme for a surface is **Map**.

When using the GUI, the **Scheme** can be set by pushing one of the highlighted radio buttons, **Map**, **Pave**, **Submap**, or **Triangle**. When using the command line, the scheme is set with the command

```
surface <range> scheme {map | pave | submap | triangle | trimap | tripave}
```

Mapping

The surface mapping capability in CUBIT is based on standard mapping transformations [7]. The transformations work well and yield high quality meshes in regions with roughly parallel opposing sides. The surface may have any number of curves defining the sides and still produce a nice mapped mesh. Figure 5-8a illustrates a mapped mesh on a Bezier surface using two biased and two equal curve meshes on the region’s boundary.

Paving

Paving (see reference [1]) allows the meshing of an arbitrary three-dimensional region with quadrilateral elements. The paver supports interior holes and arbitrary boundaries. It also allows for easy transitions between dissimilar sizes of elements. Figure 5-8 shows the same surface

meshed the mapping (left) and paving (right) schemes using the same discretization of the boundary curves.

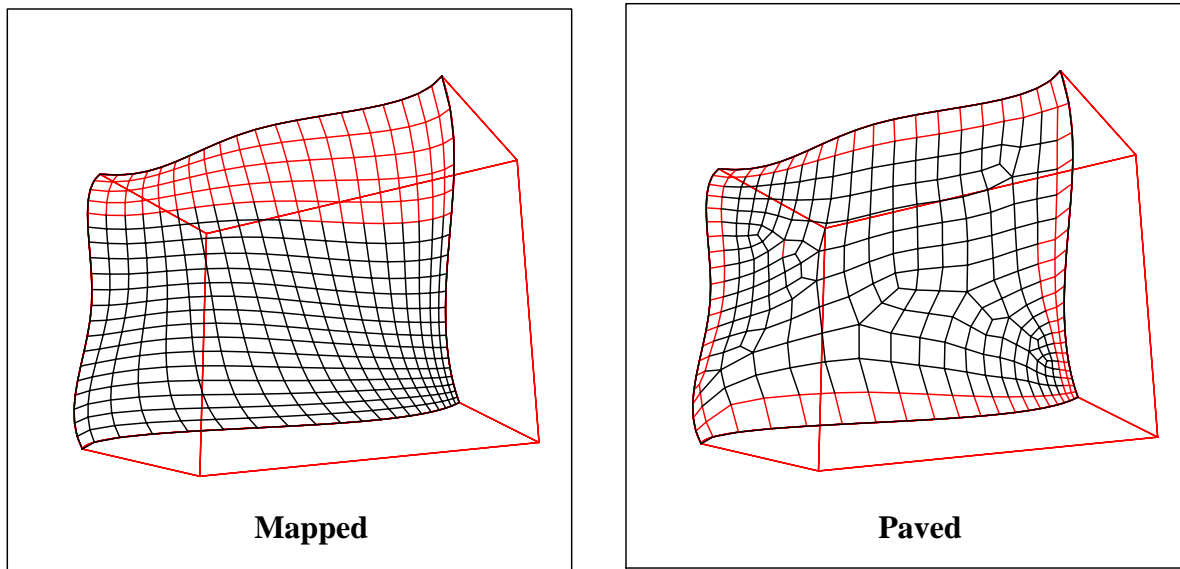


Figure 5-8 Mapped and paved surface meshing

Submapping

Submapping is a meshing tool based on the surface mapping capability discussed above. This tool is suited for mesh generation on surfaces which can be decomposed into mappable subsurfaces. This algorithm uses a pseudo-decomposition method to break the surface into simple mappable regions. Submapping is not limited by the number of corners or reversals in the geometry or by the number of edges. The submap tool, however is best suited for surfaces that are fairly blocky. If a surface geometry does not contain corners and reversals the surface will still be meshed. However, the mapping scheme may produce a better mesh.

An illustration of a mesh produced by the submapping algorithm is shown in Figure 5-9. The left side of this figure shows the topology assumed by the submapping algorithm and the right side shows the resulting mesh and the vertex classifications. An alternative interpretation of the topology is shown in Figure 5-10..

Meshing Primitives

Several basic shapes can be meshed as primitives using an internal decomposition technique to decompose the shape into mappable segments. The triangle primitive is currently the only surface meshing primitive available in CUBIT.

• Triangle Primitive

The **triangle** scheme indicates that the region should be meshed as a triangle. The definition of the triangle is general in that surfaces containing 3 natural corners can often be meshed successfully with this algorithm. For instance, the surface of a sphere octant is handled nicely by the triangle primitive. The algorithm requires that there be at least 6 intervals (2 per side) specified on the curves representing the perimeter of the surface and that the sum of the intervals on any two of the triangle's sides be at least two greater than the number of intervals on the remaining side. Figure 5-11 illustrates a triangle mesh on a 3D surface.

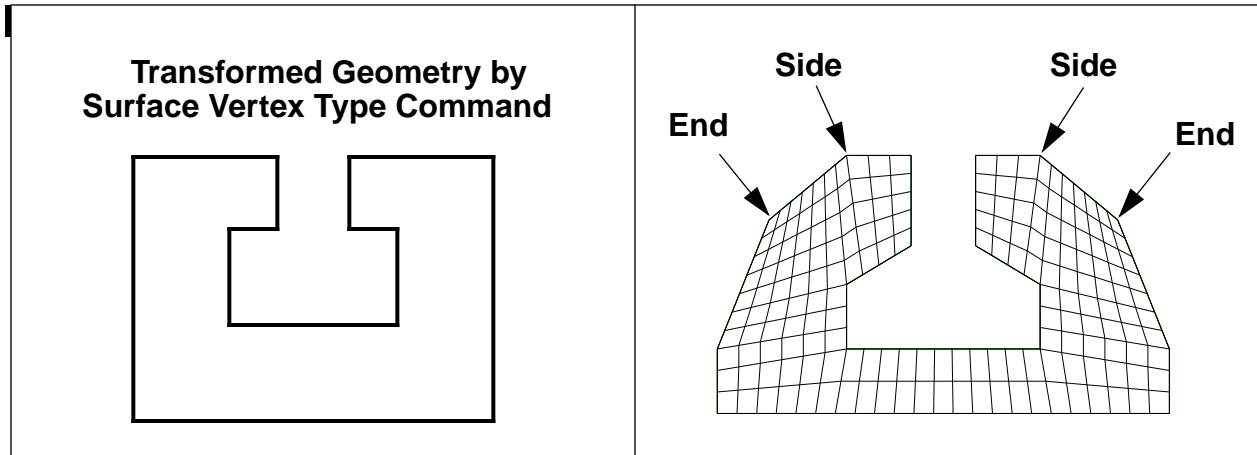


Figure 5-9 Submapping Example

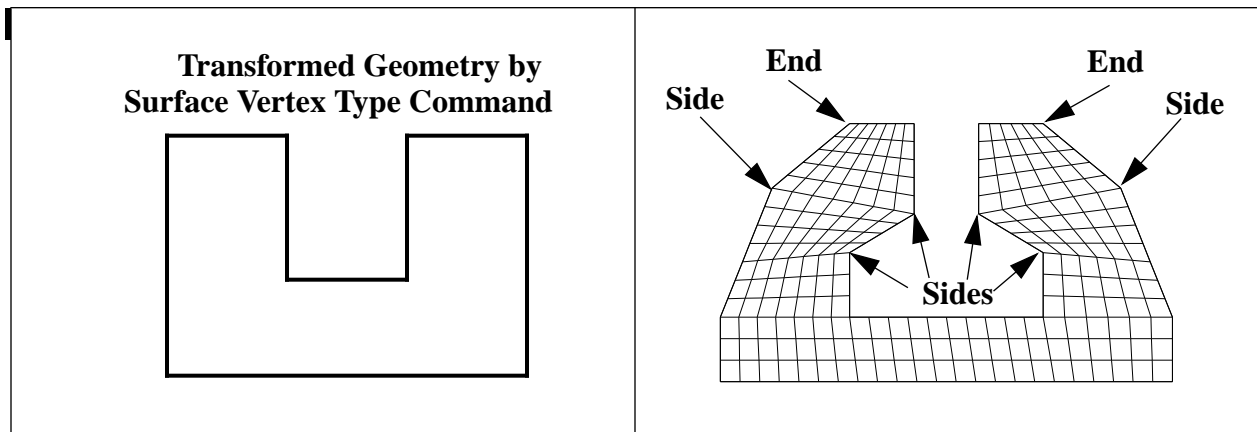


Figure 5-10 Alternate Submapping Topology Interpretation

Adaptive Surface Meshing

Adapting surface meshing in CUBIT provides smaller faces where the function value is small and vica versa, thus allowing a the mesh on a surface to adapt to some type of sizing function. Adaptive surface meshing is done using the paving algorithm in combination with an appropriate sizing function. The types of sizing functions that can be used are linear, curvature, test, and Exodus-based field function. These are each described in the following paragraphs.

The **Curvature** sizing function determines element size based on parametric curvature of a surface at the current location. Two cylinders with different radii and therefore different curvature were meshed using this sizing function; the result is shown in Figure 5-12.

The linear class of sizing functions determines element size based on a weighted average of edge lengths for mesh edges bounding the surface being meshed. There are several variants of this class of sizing function. The **Linear** function bases edge length at a location on the lengths of edges bounding the surface weighted by their inverse distance from the current location. The **Interval** function is similar to the **Linear** function, but uses the square of the edge length instead. The **Inverse** function computes inverse edge lengths instead of edge lengths. A comparison of meshes computed using the **Linear**, **Interval** and **Inverse** adaptive sizing functions to a normal paved mesh is shown in Figure 5-13.

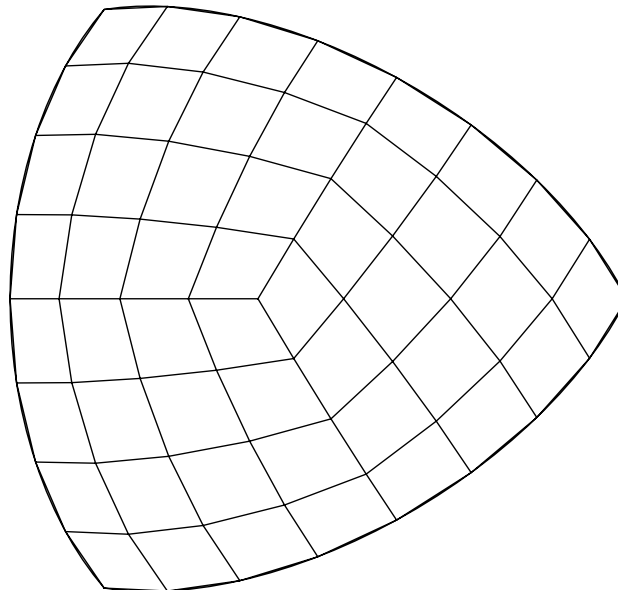


Figure 5-11 Triangle primitive mesh

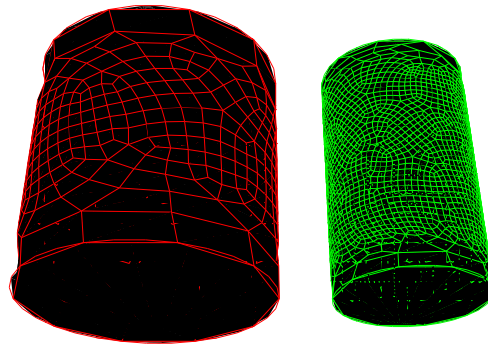


Figure 5-12 Curvature sizing function meshes on cylinders with varying radii.

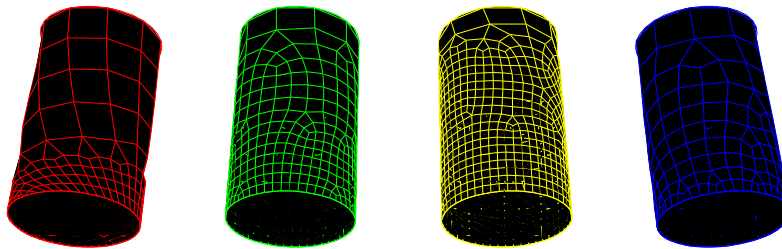


Figure 5-13 Illustration of No Sizing, Linear, Interval, and Inverse Sizing Functions

The **Test** sizing function is a hardwired numerical function used to demonstrate the transitional effect of sizing function-based and adaptive paving. The current function is implemented in the `RefFace::test_sizing_function` method in CUBIT source code, and is a periodic function which is repeated in 50x50 unit intervals on a surface. A mesh which was generated using a periodic function which is repeated in 50x50 unit intervals on a surface is shown in Figure 5-15.

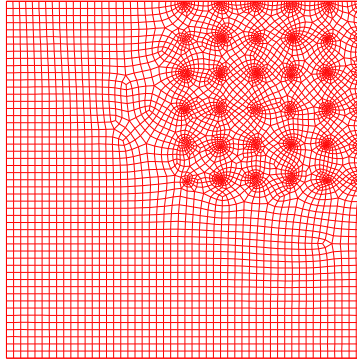


Figure 5-14 Test sizing function mesh.

The procedure for adaptively meshing a surface is to designate paving as the mesh scheme for that surface, assign sizing function types, and mesh the surface. The command syntax is:

Surface < id > Scheme Pave

Surface < id > Sizing Function Type { Curvature | Linear | Interval | Inverse | Test | Exodus }

Mesh Surface <id>

The **Exodus** sizing function type is discussed in the following subsection.

Sizing Function-Based Surface Meshing

The ability to specify the size of elements based on a general field function is also available in CUBIT. With this capability the desired element size can be determined using a field variable read from a time-dependent variable in an Exodus file. Either node-based or element-based variables can be used; for details on how to read in Exodus information to be used with adaptive paving, see <>. Importing a field function, associating the field function with a surface, and normalizing that function are done in two separate steps, to allow renormalization without having to read the mesh in again. Currently, field functions are imported from element-based ExodusII data. Thus, a field function is a time-dependent element variable in an ExodusII file. The mesh block containing the corresponding elements must be imported along with the field function data. For details on the adaptive paving algorithm, see [Ref].

Exodus variable-based adaptive paving is accomplished in CUBIT in several steps:

- Surface mesh scheme set to Pave. Bounding curve mesh schemes can also optionally be set to Stride.
- An Exodus mesh and time-dependent variable for that mesh is read into CUBIT.
- The mesh and variable data are associated to geometry.
- The Exodus variable is normalized to give localized size measures, and the surface sizing function type is designated.
- Surface is meshed.

The assignment of surface and curve mesh schemes are discussed in [ref's].

The following command is used to read in a field function and its associated mesh:

**Import Sizing Function 'exodusII_filename' Block <block_id>
Variable '<variable_name>' Time <time_val>**

where **<block_id>** is the element block to be read, **<variable_name>** is the Exodus time-dependent variable name (either element-based or nodal-based), and **<time_val>** is the problem time at which the data is to be read. When this command is given, the nodes and elements for that element block are read in and associated to geometry already initialized in CUBIT (for information on associating mesh to geometry, see [ref]). Note that when a sizing function is read in, the mesh is stored in an ExodusMesh object for the corresponding geometry, and therefore the geometry is not considered to be meshed. Also, the geometry to which the mesh is being associated must be in the same state as it was when that mesh was written (see “Import Mesh” on page 119 for more details on importing meshes).

Once the field function has been read in and assigned to a surface, it can be normalized before being used to generate a mesh. The normalization parameters are specified in the same command that is used to specify the sizing function type for the surface. The syntax of this command is:

Surface < id > Sizing Function Type Exodus
[Min <min_val> Max <max_val>]

If normalization parameters are specified, the field function will be normalized so that its range falls between the minimum and maximum values input. Subsequent normalizations operate on the normalized data and not on the original data. If an element-based variable is used for the sizing function, each node is assigned a sizing function that is the average of variables on all elements connected to that node. Nodal variables are used directly.

After the sizing function normalization, the surface can be meshed using the normal meshing command

Mesh Surface <id>

For example, Figure 5-15 depicts a plastic strain metric which was generated by PRONTO-

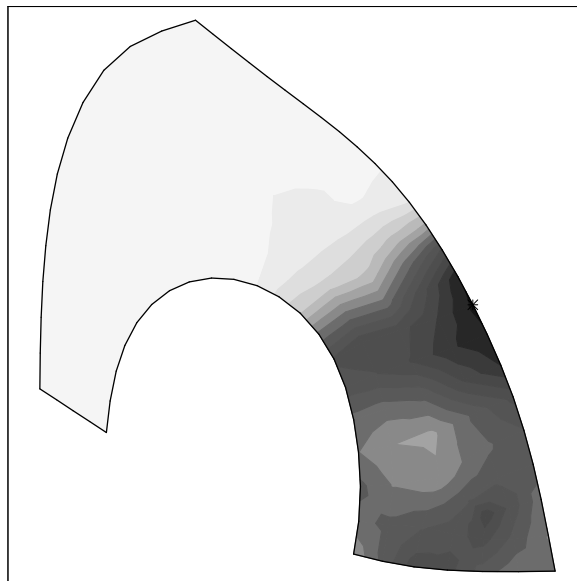


Figure 5-15 Plastic strain metric

3D [18], a transient solid dynamics solver, and recorded into an ExodusII data file. When the file is read back into CUBIT, the paving algorithm is driven by the function values at the original

node locations, resulting in an adaptively generated mesh [19]. Figure 5-16 depicts the resulting

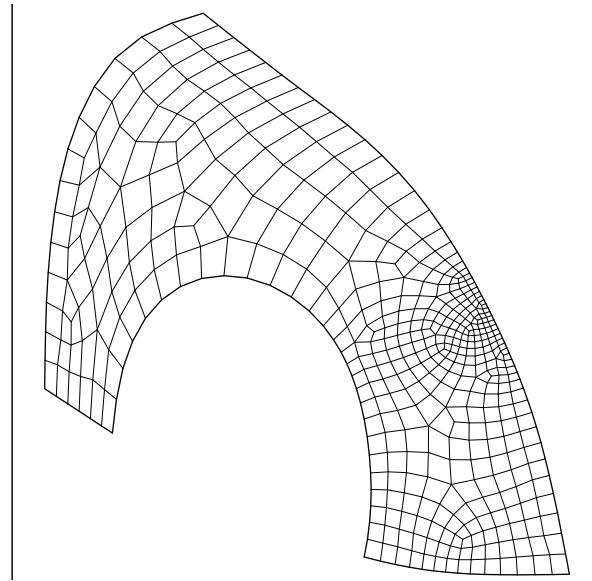


Figure 5-16 Adaptively generated mesh

mesh from this plastic strain objective function.

Boundary Layer Tool

The Boundary Layer meshing scheme is an algorithm designed to insert rows of elements around the boundary of a surface before meshing the interior. The aspect ratios of these elements can be carefully controlled. This capability is specifically designed for fluid simulations involving a boundary layer. With this tool, the total boundary layer thickness and relative thicknesses of each of the rows can be specified.

A boundary layer is specified as a set of parameters, with a boundary layer ID. This set of parameters is then attached to curve/surface pairs in the geometry. Thus a curve may have a row of boundary layer elements next to it on one of the surfaces it bounds, but not on another. A boundary layer is currently defined using a combination of four of the five possible parameters. These parameters are shown in Figure 5-17.

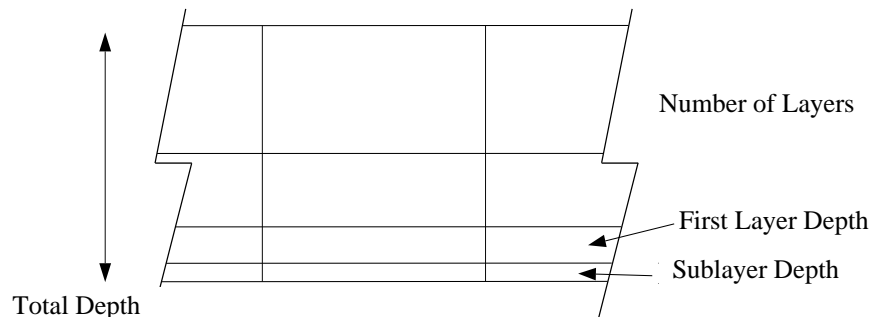
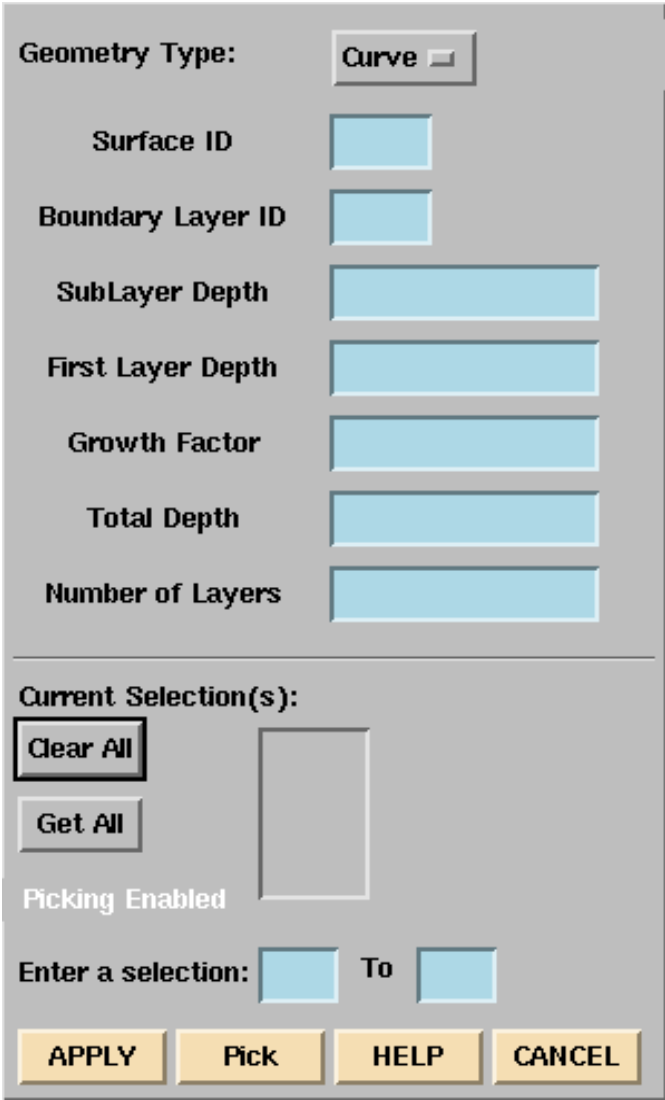


Figure 5-17 Boundary Layer Parameters

- **First Layer Depth.** This parameter specifies the physical depth of the first layer of elements. This is a required parameter.

- **Growth Factor.** This parameter specifies the relative difference of each subsequent layer's depth. For instance, a factor of 1.2 makes each layer 1.2 times the preceding layer's depth. This is an required parameter unless the second layer depth is specified.
- **Total Depth.** This parameter specifies the total depth of all the boundary layers. Either the total depth or the number of layers must be specified.
- **Number of Layers.** This parameter specifies the total number of layers in the boundary layer. Either the number of layers or the total depth must be specified.
- **Sublayer Depth.** This parameter specifies the physical depth of the sublayer of elements. This is an optional parameter. If it is specified a sublayer of elements (not counted in the "number of layers" parameter) is added.

When using the GUI, the **Boundary Layer** menu item under the Mesh menu will display the dialog shown in Figure 5-18. The **Boundary Layer ID** sets the boundary layer either to be



The dialog box is titled "Boundary Layer" and contains the following fields and controls:

- Geometry Type:** A dropdown menu currently set to "Curve".
- Surface ID:** A text input field.
- Boundary Layer ID:** A text input field.
- SubLayer Depth:** A text input field.
- First Layer Depth:** A text input field.
- Growth Factor:** A text input field.
- Total Depth:** A text input field.
- Number of Layers:** A text input field.
- Current Selection(s):** A section containing:
 - Clear All** and **Get All** buttons.
 - A large empty rectangular box for displaying selections.
 - Picking Enabled** checkbox.
 - Enter a selection:** and **To** text labels, each followed by a small text input field.
- Buttons:** **APPLY**, **Pick**, **HELP**, and **CANCEL** buttons at the bottom.

Figure 5-18 Boundary Layer Dialog Box

defined or attached to a curve/surface pair. Specifying the **Surface ID** and **Curve ID** sets the

curve/surface pair for the application of the boundary layer. The **Curve ID** is set in the picker window at the bottom of the dialog. If the user only wishes to define a boundary layer, and not apply it to any particular curve/surface pair, these two parameters need not be filled in. The **SubLayer Depth**, **First Layer Depth**, **Growth Factor**, **Total Depth**, and **Number of Layers** are the parameters that may be filled in to define a new boundary layer. These must be supplied in the combinations described above. If the user is simply attaching an existing boundary layer to a curve/surface pair, these parameters should not be used. Pushing the **Apply** button performs the definition and/or attachment of the boundary layer to the curve/surface pair.

When using the command line, a boundary layer is set with the command

**BoundaryLayer <range> First [Layer] <depth> Growth [Factor] <growth>
Total Depth <depth> [Sublayer <depth>]**

**BoundaryLayer <range> First [Layer] <depth> Growth [Factor] <growth>
Layers <count> [Sublayer <depth>]**

Note: If the growth factor, total depth, and number of rows are specified together, then the boundary layer definition is overspecified, and the total depth will be ignored.

The boundary layer is attached to curve/surface pairs with the command:

BoundaryLayer <layer_id> Surface <range> Curve <range>

Meshing the Surface

Once the desired scheme has been chosen, and any boundary layers for the surface defined and attached, the surface can be meshed. If the user wishes to receive a different element type than the default (four node quads) for surface meshing, this specification needs to be set prior to creating any surface meshes. If using the GUI, the **Mesh Now** button on the **Mesh Control** dialog box shown in Figure 5-3 is pushed to mesh the surface(s).



Note: *The **Apply** button must be pushed before the settings shown in the dialog are stored on the geometry and used when the **Mesh Now** button is pushed.*

If using the command interface, the appropriate command is:

mesh surface <range>

The resulting mesh will be drawn on the screen in the mesh color designated for the surface.

▼ Volume Meshing

Volume meshing discretizes the volume into nodes, edges, faces, and hexahedral elements. When meshing a volume, the bounding surfaces of the volume are first meshed (if not already meshed). Available volume meshing algorithms are **mapping**, **project**, **translate**, **rotate**, **plaster**, and **whisker weaving** (**plaster** and **whisker weaving** are currently under development and as such are not recommended for production use). When using the GUI, the Mesh Control option of the **Mesh** menu button will produce the dialog box shown in Figure 5-19, which when **Geometry Type** is set to **Volume** can be used to control volume meshing. Node density and relative spacing along the curves which bound the surfaces of the volume can be set as explained in “Curve Meshing” on page 66.

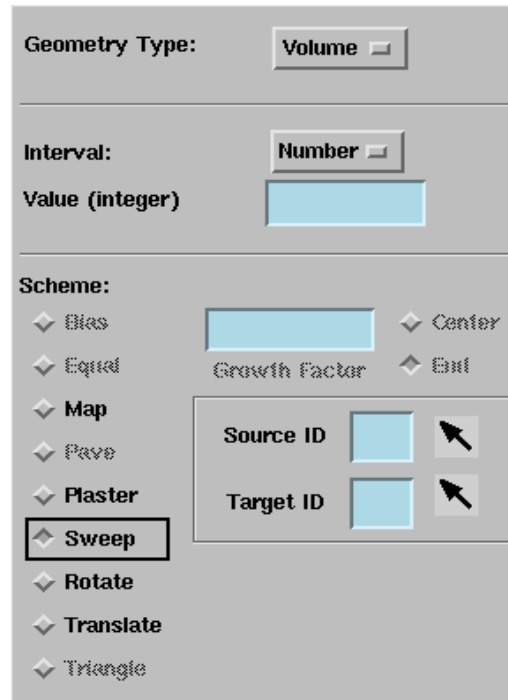


Figure 5-19 Volume Meshing with the GUI Mesh Dialog

Scheme Designation

The algorithm to be used for volume meshing is designated as the scheme of the volume. Currently, valid schemes are:

Map Create mesh using mapping transformations.

Project Create mesh by projecting the mesh from one surface to another.

Translate Create mesh by translating the mesh from a source surface along the vector from the source surface to the target surface.

Rotate Create mesh by rotating about an axis from the source surface to the target surface.

Plaster Fill the volume in a free meshing inward approach—currently being researched.

Weave Attempt to generate whisker weaving sheets to fill the volume—currently being researched.

Each of these algorithms are briefly described below. The default scheme for a volume is **Map**. When using the GUI, the **Scheme** can be set by pushing one of the highlighted radio buttons, **Map**, **Project**, **Translate**, **Rotate**, **Plaster**, or **Weave**. When using the command line, the scheme is set with the commands

```
volume <range> scheme {map | weave | plaster}
```

```
volume <range> scheme {project | translate | rotate} Source <id> Target <id>
```

Mapping

The volume mapping capability in CUBIT is based on the standard mapping transformations [7] discussed in the surface meshing section. Volume mapping is designed to work on volumes that are in some sense a logical cube (they have 6 logical surfaces and 8 logical vertices). There may

be more or less actual surfaces, as long as the logical surfaces can be determined. For example the union of two blocks shown in Figure 5-20 contains 8 surfaces, but it is easy to see that 4 side

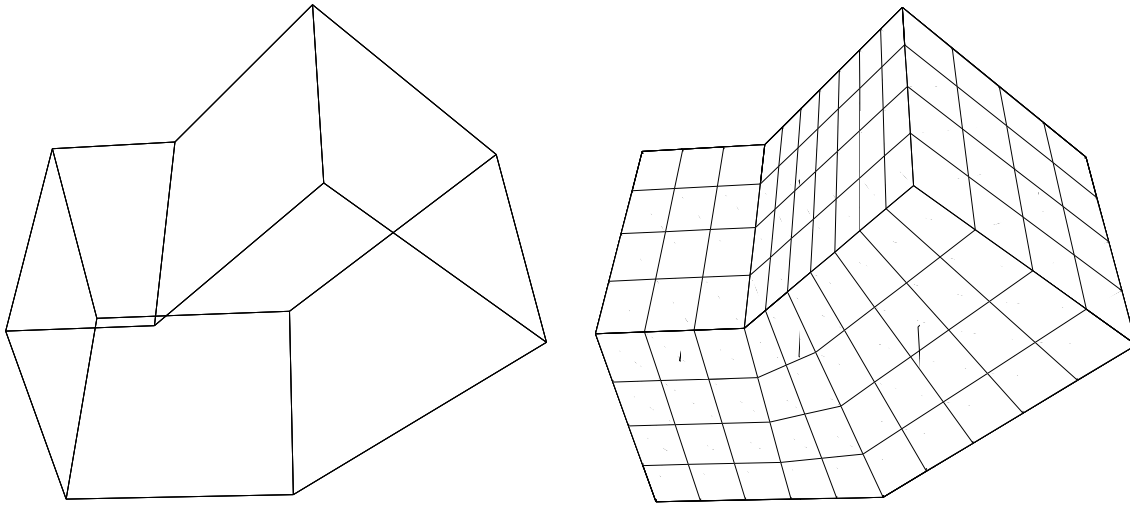


Figure 5-20 Volume Mapping of an 8-Surfaced Volume.

surfaces can be logically combined to form 2 surfaces of the logical cube and mapping can be performed successfully. On the other hand, the quarter cylinder shown in Figure 5-21 has only

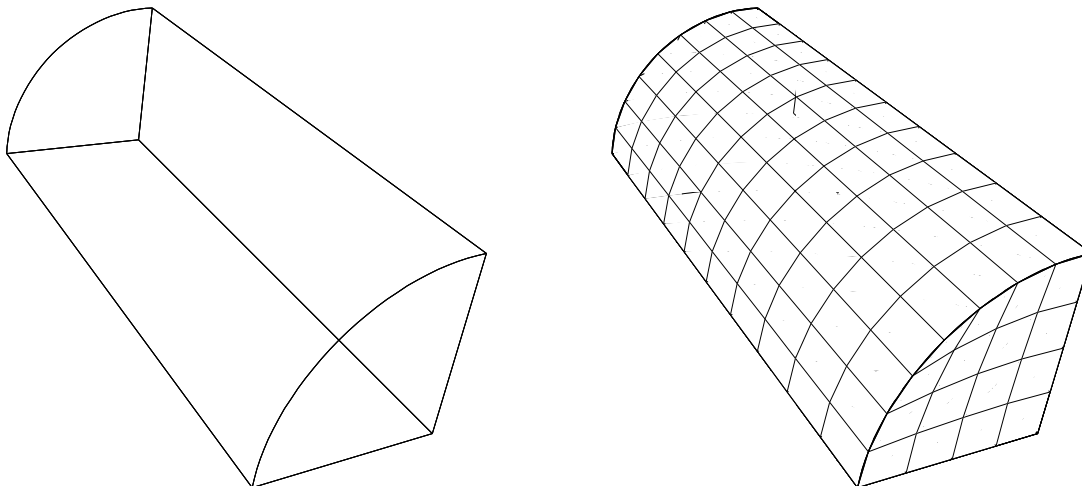


Figure 5-21 Volume Mapping of a 5-Surfaced Volume

5 surfaces. However, the cylindrical surface can be logically dissected to form 2 of the logical surfaces, and this volume can also be meshed successfully. The volume mapper in CUBIT needs no input from the user to determine which of the surfaces need logical dissection and/or combination. The surface mesh, as described below, dictates these choices.

The pattern of the surface mesh will dictate whether a volume can be mapped. On any mappable volume mesh, the surface mesh must contain only 8 trivalent nodes (nodes attached to only 3

quadrilateral faces on the surface). All other nodes must be quadvalent (4 elements attached to the node). These 8 trivalent nodes form the corners of the cube to be mapped. In Figure 5-22

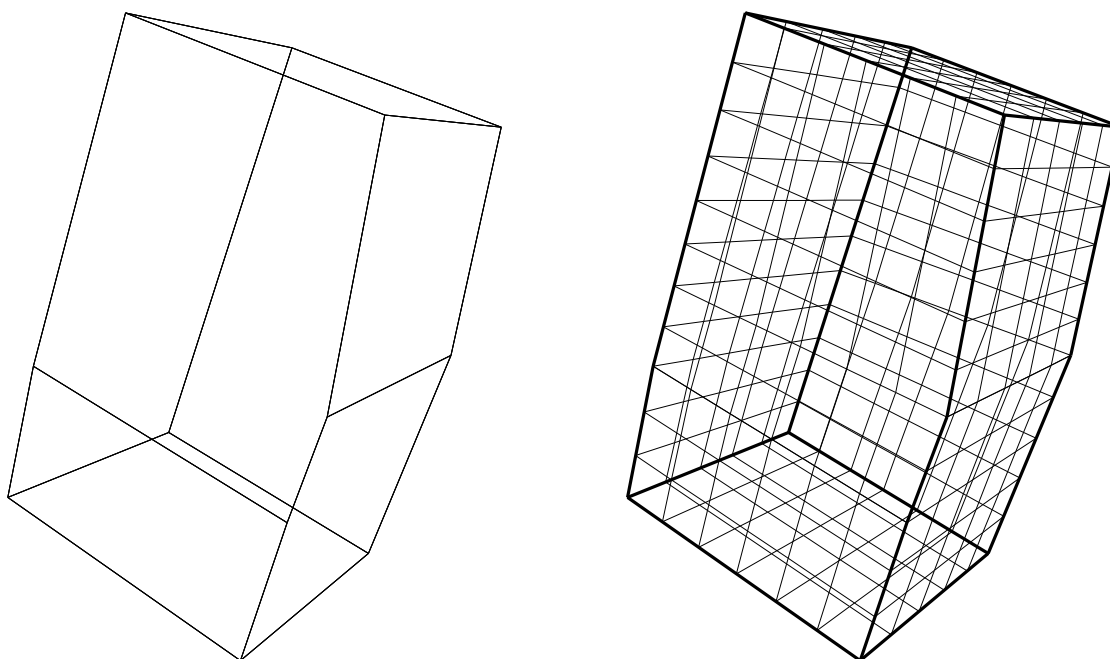


Figure 5-22 Surface Mesh of an 8-Surfaced Volume Highlighting the Logical Edges Used For Volume Mapping.

the surface mesh of an 8-surfaced volume is shown. The logical edges of the surface mesh are highlighted. Notice that not all the geometric edges are used as logical edges for meshing. These logical edges meet at the trivalent nodes of the surface mesh (the corners). This combination of 8 trivalent and the rest quadvalent nodes on the surface can only produce a logical cube. Thus, the user need only insure that the surface mesh has the right characteristics for volume mapping to succeed.

Sweeping (Project, Translate, and Rotate)

The CUBIT volume sweeping capability is divided into three algorithms (Project, Translate, and Rotate) which each generate a volume mesh by extruding hexahedrons from a previously meshed source surface to a topologically similar target surface. Topologically similar includes relationships such as a rectangular surface being extruded into an elliptical surface as long as the two surfaces contain the same number of boundaries or loops¹.

The geometric requirements for a sweeping operation are that the volume be “2 and 1/2 D,” or extrudable. This requirement is typically satisfied if the surfaces linking the source surface and target surface can be meshed with compatible mapping transformations (See “Mapping” on page 99.), where the “compatible” qualifier means that the edges linking the source surface to the target surface have the same number of intervals. The source surface may be meshed using any of the meshing methods described in “Surface Meshing” on page 98. The smoothed

1. The number of loops on a surface refers to the number of boundaries it has. A surface always has at least one boundary, the set of curves which bound it externally. Some surfaces also have internal boundaries, or loops, in the form of holes.

topology of the source surface mesh will be reproduced on the target surface unless the target surface is already meshed. In this case, the target surface mesh must have the same topology and connectivity as the source target mesh. It is much more efficient to let the sweeping meshing algorithms mesh the target source face if at all possible.

The procedure for the sweeping volume mesh generation algorithms is as follows: first the attributes (interval settings, element type, etc.) of the volume should be set, and then the surfaces which will act as the source and target must be selected. When the command to mesh the volume is executed, the sweeper will mesh the source surface, and then the linking surfaces. The sweeping algorithms will then project a layer at a time, progressing through the unmeshed volume. The difference between the three sweeping algorithms is the method used to project the nodes and elements from one layer to the next. These details will be discussed in the following sections.

• **Project**

The **project** sweeping algorithm is a modified version of the plastering hex element projection. The sweep path can be completely general. The nature of the swept region can also be general in that it can contain draft angles and non-symmetric transformations. Figure 5-23 displays swept meshes involving mapped and paved source surfaces. The project algorithm can also

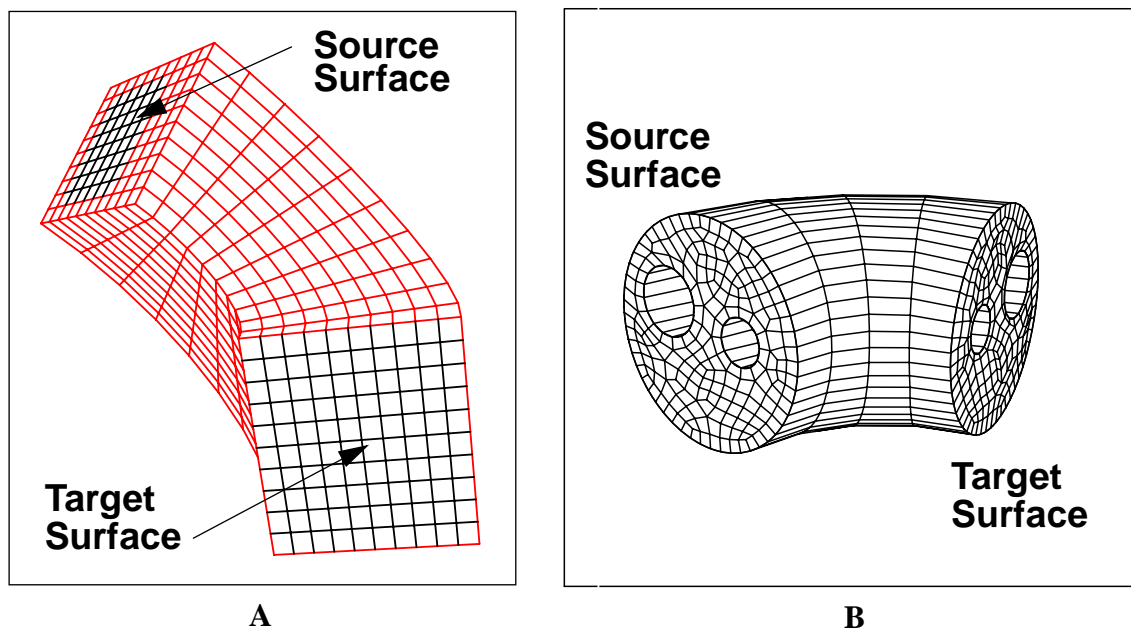


Figure 5-23 Project Volume Meshing

handle multiple surfaces linking the source surface and the target surfaces. An example of this is shown in Figure 5-24. Note that for the multiple surface meshing case, the interval requirement is that the total number of intervals along each multiple edge path from the source surface to the target surface must be the same for each path.

The project algorithm proceeds by determining a “projection node” for each node on the boundary of the current “layer.” A node’s projection node is the node directly “above” (in the sense of up being closer to the target surface than the source surface). An approximate planar surface is then generated through these nodes. For each node interior to the boundary of the

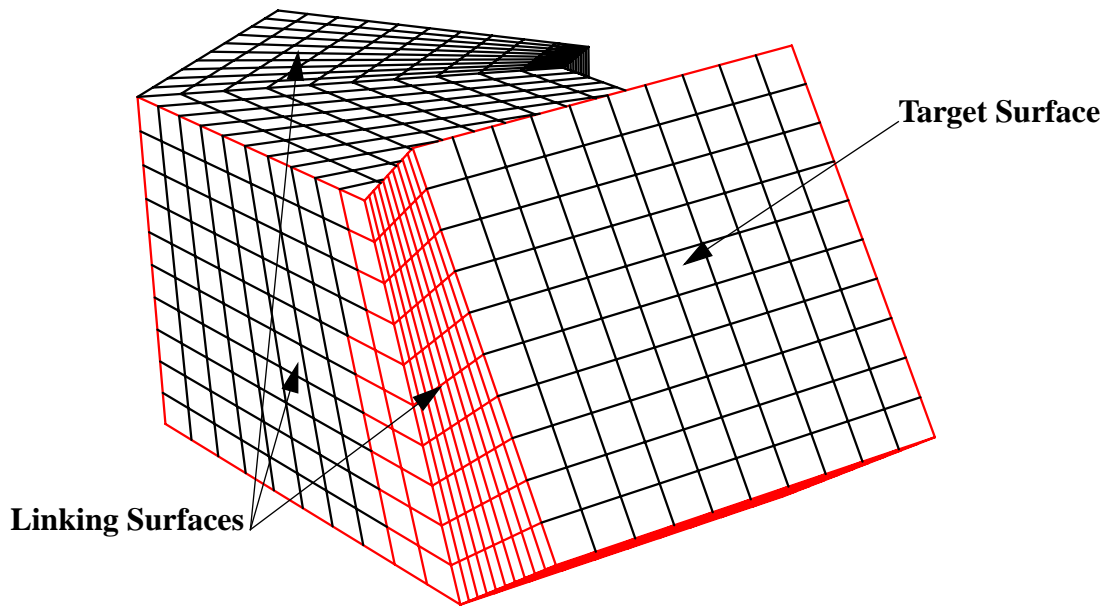


Figure 5-24 Multiple Surface Project Volume Meshing

volume, an average “projection vector” is calculated by determining which neighboring nodes have existing projection nodes and averaging the vector from these nodes to their projection nodes. The interior nodes are then calculated by projecting from the interior nodes along the average projection vector to the approximate planar surface calculated earlier. The interior nodes are ordered in a manner to maximize the number of neighboring nodes with existing projection nodes. This process is repeated for each interior node on the current layer. After all projection nodes have been created, a new layer of hexes is created and smoothed. The process then repeats for the next layer.

If the approximate planar surface does not closely match the surface defined by the boundary projection nodes, the interior projection nodes are created simply by projecting along the average projection vector; the intersection with the planar surface is not calculated.

The project algorithm is very general in that it can create a mesh on almost any extrudable volume; however, this generality has some disadvantages in that it does not use any global information about the actual generation of the volume. It simply projects a layer at a time in moving from the source surface to the target surface. Because of this, and the smoothing that is done after each layer is created, features that are present in the source surface mesh sometimes tend to get smoothed out or smeared by the time the mesh reaches the target surface mesh. The project algorithm is also slower and requires more memory than the translate and rotate algorithms since it must calculate a local projection for each node and maintain the information required by the smoothing algorithms.

• **Translate**

The **translate** sweeping algorithm is a more restricted version of the **project** algorithm. It is used when the source surface and target surface have exactly the same geometry and are parallel. If it is possible to translate the source surface along a vector and have it completely overlay the target surface, this algorithm can be used.

The **translate** algorithm proceeds by calculating the vector from the source surface to the target surface. The thickness of each layer is then calculated as the distance from a boundary node on

that layer to that node's projection node. This distance is the same for each node on the current layer since the source surface and the target surface (and therefore, each layer) are parallel. Each interior node is then projected that thickness along the vector. This process is repeated for each layer in the volume. No smoothing is performed on the generated volume mesh.

• **Rotate**

The **rotate** sweeping algorithm is also a restricted version of the **project** algorithm. It is used when the source surface and target surface are exactly the same and are connected by a conic or toroidal surface. If it is possible to rotate the source surface about a single axis and have it completely overlay the target surface, this algorithm can be used. This algorithm cannot be used if the rotation axis contacts either the source surface or the target surface, that is, there must be a hole through the center of the generated mesh.

The **rotate** algorithm proceeds by calculating the axis of rotation from the source surface to the target surface. The thickness of each layer is calculated from the amount of rotation from a boundary node on that layer to that node's projection node. This rotational distance is the same for each node on the current layer. This process is repeated for each layer in the volume. No smoothing is performed on the generated volume mesh.

Plastering

Plastering uses the discretized surface and begins to lay elements into the interior of the volume. This continues until the volume fills, with adjustments made to the exterior surface mesh as deemed necessary. This algorithm is currently under development and not suggested for use although it may be tested if desired. It should currently perform well for blocky structures where the surface mesh will form a valid boundary for an interior hex mesh. Some examples of these structures are shown in Figure 5-25. These structures allow very straightforward hex element

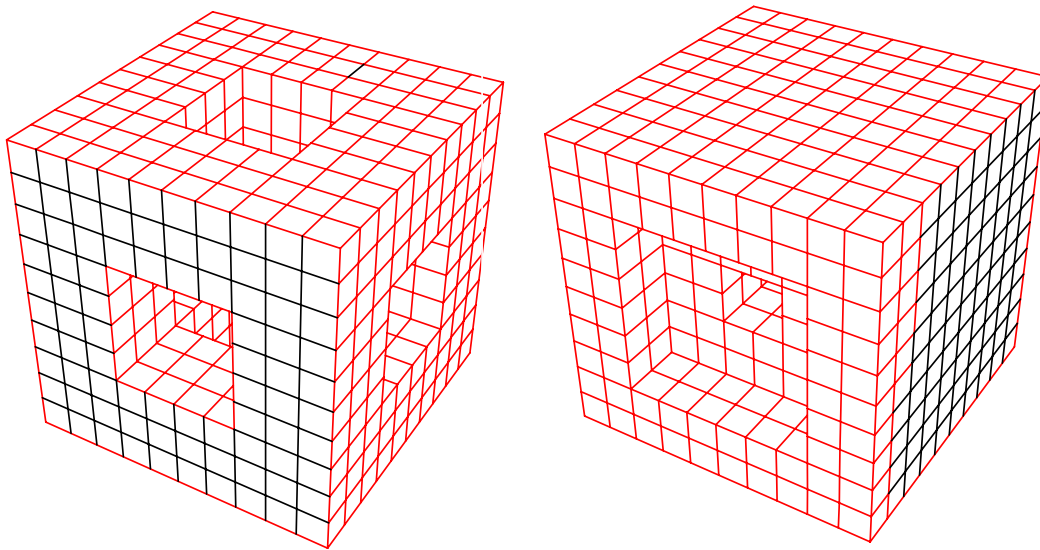


Figure 5-25 Plastering Examples

connectivity and do not contain any irregular nodes (nodes that are shared by other than four element edges in a given layer).

Whisker Weaving

Whisker weaving is based on information contained in the Spatial Twist Continuum (STC), which is the geometric dual of an all-hexahedral mesh. Whisker weaving begins with a three dimensional geometry and an all-quadrilateral surface mesh, then constructs hexahedral element connectivity from the boundary inward. The result of the whisker weaving algorithm is a complete representation of hex mesh connectivity. This connectivity is then converted into an actual mesh and smoothed to fit the volume.

The whisker weaving algorithm is a relatively new meshing technique, and as such is capable of meshing only simple volumes at this time. Examples of meshes generated using the whisker weaving algorithm are shown in Figure 5-26.

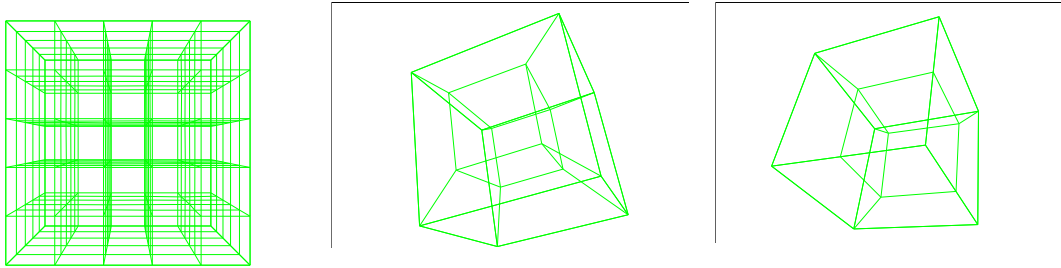


Figure 5-26 Whisker Weaving meshes.

Meshing the Volume

Once the desired scheme has been chosen, the volume can be meshed. If the user wishes to receive a different element type than the default (eight-node hexahedrons) for volume meshing, this specification needs to be set prior to creating any volume meshes. If using the GUI, the **Mesh Now** button is pushed to mesh the curve(s).



Note: The **Apply** button must be pushed before the settings shown in the dialog are stored on the geometry and used when the **Mesh Now** button is pushed.

If using the command interface, the appropriate command is:

mesh volume <range> | All

mesh body <range> | All

The resulting mesh will be drawn on the screen in the mesh color designated for the volume.

▼ Mesh Duplication

If the geometry to be meshed was generated using the body copy command explained in “Copy Bodies” on page 52, then the mesh from the original geometry can be copied directly to the new geometry using the command

copy mesh {volume|surface} <id1> onto {volume|surface} <id2>

This functionality is not yet available in the GUI version of CUBIT.

▼ Mesh Editing

A limited capability to modify portions of a mesh is provided to allow the user to make judgements about the level of smoothing required for volumetric meshes. Since the models to be meshed vary widely, and since CUBIT does not contain a sophisticated geometry recognizer, CUBIT is unable to make decisions regarding the type of smoothing algorithm to employ. Certain algorithms work well for classes of problems and fail for others. There seems to be no perfect smoothing algorithm, therefore the decision of which type of smoother to use is left to the user.

Smoothing

Surface

Surface smoothing algorithms currently consist of a variety of equipotential stencils, length-weighted laplacian, and centroid area pull. The nature of equipotential smoothers is one of weight equalization between adjacent nodes. For a generic, area, or Jacobian-based weighted smooth, this is roughly similar to equalizing areas between adjacent elements. The techniques behave well for regular or irregular grids on non-periodic surfaces, but are not yet released for periodic surfaces such as cylinders, spheres, (some) nurbs, and tori.

The Laplacian smoothing approach calculates an average element edge length around the mesh node being smoothed to weight the magnitude of the allowed node movement [8]. Therefore this smoother is highly sensitive to element edge lengths and tends to average these lengths to form better shaped elements. However, similar to the mapping transformations, the length-weighted Laplacian formulation has difficulty with highly concave regions⁸.

The Centroid Area Pull smoothing approach attempts to create elements of equal area by. Each node is pulled toward the centroids of adjacent elements by forces proportional to the respective element areas [8].

Smoothing is implemented like the meshing, where the scheme is set first and the action performed later, with separate commands. The command line syntax for setting the smoothing scheme for a surface is as follows:

**Surface <range> Smooth Scheme Equipotential [Fixed]
[Weight {Jacobian | Area | Inverse [Area]}]**

Surface <range> Smooth Scheme Laplacian [Fixed]

Surface <range> Smooth Scheme Centroid Area Pull [Fixed]

If the **Weight** keyword is not specified, a **Generic** weighting is used by default. The **Fixed** keyword forces the nodes lying on the bounding curves of a surface to remain stationary instead of “floating” along the equation of the curve until all nodes have converged. Note that this restriction limits the amount of impact the smoothing operation can have on the surface mesh.

To smooth a surface based on the previously set scheme, the following command is used:

Smooth Surface <range> [global]

If no scheme has been set, the Equipotential scheme is used by default. The optional global identifier is only valid with the laplacian and centroid area pull smoothing schemes. If entered, all surfaces specified by range will be smoothed at one time. If global is not specified, the surface will be smoothed sequentially.

Volume

Two volume smoothing algorithms are currently available in CUBIT. The first type of user controlled smoother is the length-weighted Laplacian. This smoothing approach calculates an average element edge length around the mesh node being smoothed to weight the magnitude of the allowed node movement [8]. Therefore this smoother is highly sensitive to element edge lengths and tends to average these lengths to form better shaped elements. However, similar to the mapping transformations, the length-weighted Laplacian formulation has difficulty with highly concave regions⁸.

The second type of smoother is a variation of the equipotential [8] algorithm that has been extended to manage non-regular grids [9]. This method tends to equalize element volumes as it adjusts nodal locations. The advantage of the equipotential method is its tendency to “pull in” badly shaped meshes. This capability is not without cost: the equipotential method may take longer to converge or may be divergent. To impose an equipotential smooth on a volume, each element must be smoothed in every iteration—a typically expensive computation. While a Laplacian method can complete smoothing operations with only local nodal calculations, the equipotential method requires complete domain information to operate.

Smoothing is implemented like the meshing, where the scheme is set first and the action performed later, with separate commands. The command line syntax for setting the smoothing scheme for a volume is as follows:

**Volume <range> Smooth Scheme
{Laplacian | Equipotential} [Fixed]**

The **Fixed** keyword force the nodes lying on the bounding surfaces of a volume to remain stationary instead of “floating” along the equation of the surface until all nodes have converged. Note that this restriction limits the amount of impact the smoothing operation can have on the volume mesh.

To smooth a volume based on the previously set scheme, the following command is used:

Smooth Volume <range>

If no scheme has been set, the **Equipotential** scheme is used by default.

While future objectives include investigation into weighting schemes to explicitly control mesh flow according to user-defined field or element functions, a simple hex weighting function exists which demonstrates the potential of the equipotential smoothers. This command applies a user specified weight to a group of hex elements, in this case, the elements which contain a mesh face which belongs to a specified geometric surface. By adjusting the weight and running the smoother, one can expand or compact the elements being weighted. This capability may be used eventually to control adaptive hex element meshing, and to assist the free-form three-dimensional volumetric algorithms in their efforts to maintain element quality. The command syntax for applying the hex element weights is as follows:

Weight Hexes Surface <range> <weight>

Accessing Smooth Functions in the GUI

The **Smooth** menu item is located in the **Mesh** menu. The SmoothDialog for both surface smoothing and volume smoothing is shown in Figure 5-27. Each radio button corresponds to a smoothing command described in the previous two sections. The bottom portion of the SmoothDialog is a standard picker configuration described in “Picker Window” on page 59. The

geometry type, either **Surface** or **Volume**, is specified in the **Geometry Type** option menu, and specific entities of this type are selected in the PickerWindow. After selecting the **Smoothing Method**, click **Apply** to perform the smoothing or **Cancel** to abort the smoothing.

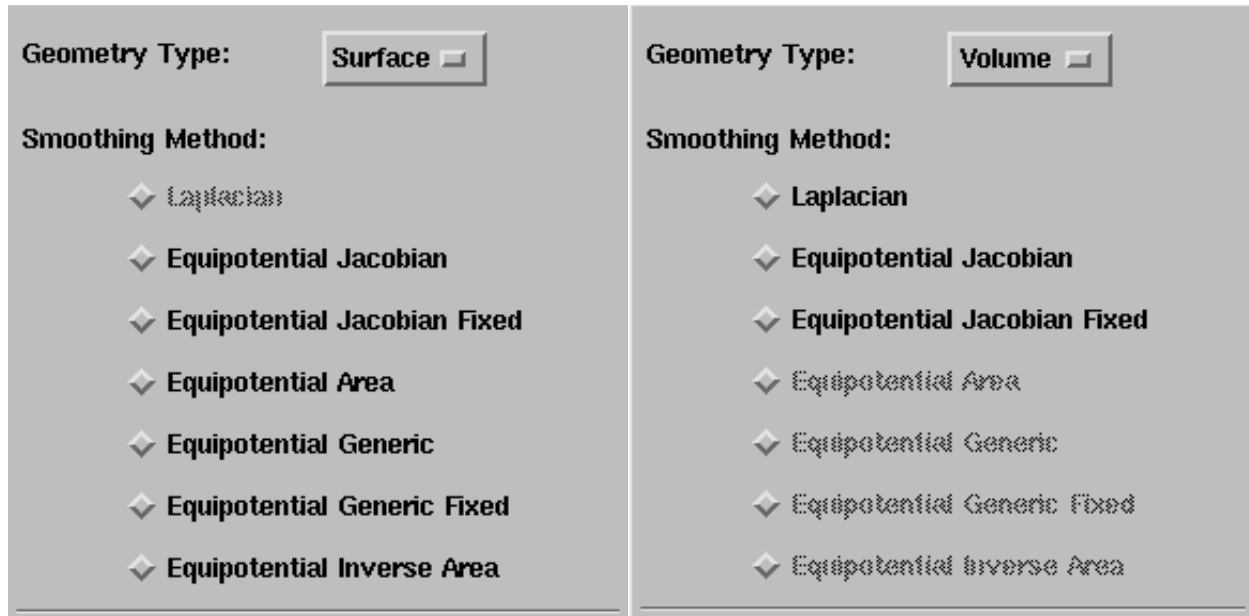


Figure 5-27 Smooth Surface and Smooth Volume Dialog Boxes

Node Repositioning

The user can reposition nodes appearing in the same nodeset using the **NodeSet Move** command. Moves can be specified using either a relative displacement or an absolute position. The command to reposition nodes in a nodeset is:

```
nodeset <id> move <delta_x> <delta_y> <delta_z>
nodeset <id> move to <x_position> <y_position> <z_position>
```

The first form of the command specifies a relative movement of the nodes by the specified distances and the second form of the command specifies absolute movement to the specified position.

Individual nodes can be repositioned using the Node Move command. Moves are specified as relative displacements. The command syntax is:

```
Node <range> Move <delta_x> <delta_y> <delta_z>
```

Mesh Deletion

The **Delete All** menu item in the **Mesh** menu permanently removes all mesh entities from the model and resets the mesh entity identification counters. A warning dialog will appear requesting a confirmation of this action. The command line syntax is:

```
delete mesh
```

Several additional commands are available through the command line version of CUBIT. Partial delete capabilities exist for volumes, surfaces, curves, and vertices. Only the mesh which is owned by or is dependent on the specified geometry will be removed.

The routines are intelligent in the respect that if a mesh delete command is executed for a surface, curve, or vertex which belongs to one (or more) fully meshed volumes, CUBIT will delete all mesh entities which must be deleted as a result: for example, if one node on the vertex is deleted, then the mesh which lies on the connected curves to that vertex is incomplete and will be deleted, and then the surface mesh which relied on the curves must also be deleted, and finally the interior hex elements within the volume. The command syntax for these commands is as follows:

delete mesh volume <range>

delete mesh surface <range>

delete mesh curve <range>

delete mesh vertex <range>

In the GUI version, selecting the **Delete** option from the **Mesh** menu will display the Mesh Delete dialog box shown in Figure 5-28. This dialog box contains the standard picker window. After the correct geometry type is specified (Vertex, Curve, Surface, or Volume), and the entity IDs are selected, either by picking or by entering the values, selecting the Apply button will display the mesh delete warning dialog. The deletion will be performed if OK is selected.

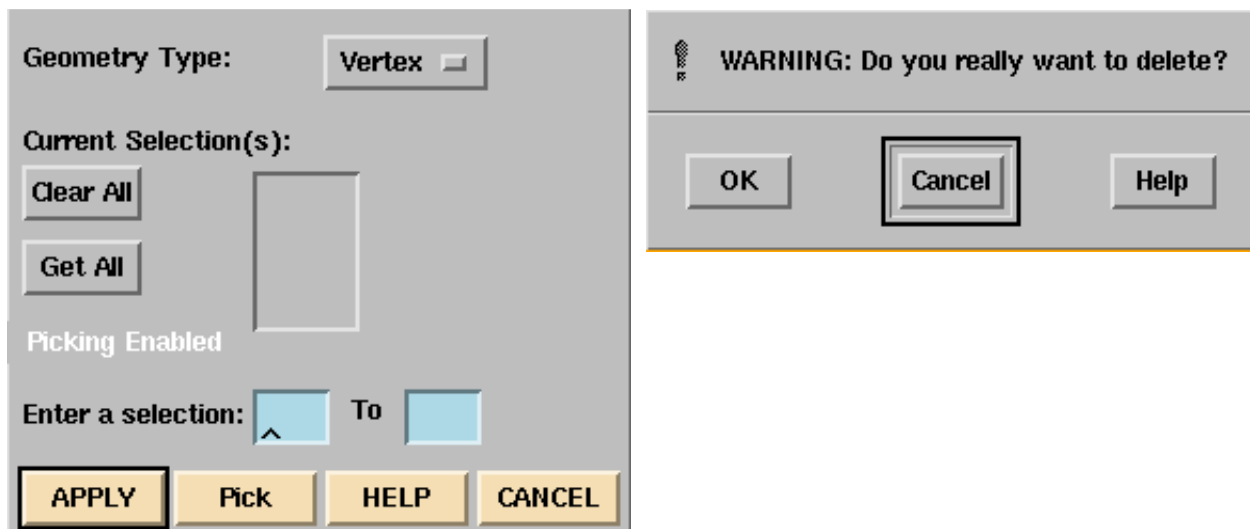


Figure 5-28 Mesh Delete and Mesh Delete Warning Dialog Boxes

Face Deletion

Mesh faces can be deleted individually using the **Delete Face** command. This command closes a face by merging two mesh nodes indicated in the input. The syntax for this command is:

Delete Face <face_id> Node1 <node1_id> Node2 <node2_id>

This command is provided primarily for developers' use, but also provides the user fine control over surface meshes. At the present time, this command works only with faces appearing on geometric surfaces and should be used before any hex meshing is performed on any volume containing the face to be deleted.

Import Mesh

A limited capability exists to read in a previously created mesh from an existing ExodusII file and associate the mesh (retaining full functionality for additional meshing or smoothing operations which depend on the imported mesh) with matching geometry in the CUBIT model. This command is useful for continuing a previous meshing problem at the point it was terminated rather than regenerating the entire mesh from the beginning. The geometry which matches the original mesh must be present in the model for the command to work.

Two methods have been implemented for associating a mesh with a geometry. The first method is proximity-based, that is it compares node positions with those of vertices, curves, etc., to determine the nodal associativity. This method has been known to fail when computing ownership on periodic curves and surfaces, and may not be robust in cases where the proximity test is not sufficiently restrictive. This method has been retained mainly for compatability with old meshes.

The second associativity method is based on logical associativity data stored in the ExodusII file. Each geometry entity has a corresponding nodeset which contains the nodes owned by that entity, and nodesets are associated back to geometry entities using geometry entity names (see “Entity Names” on page 88). (To store associativity information in the ExodusII file, the **NodeSet Associativity** command must be entered before issuing the **Export Genesis** command; see “Nodeset Associativity Data” on page 126.) Note that before importing a mesh to be associated with a geometry, the geometry must be in the same state it was in when the mesh block was written. That is, the topology must be the same, and there must be corresponding named geometry entities for each of the entities owning mesh in the original problem. This can be done either by executing the same geometry generation sequence, or by arriving at the geometry using a different method and explicitly naming the geometry entities to match those in the original problem. Only the names are used to find matching geometry entities (i.e. no check is made to ensure that the entity types match).

The same command line syntax is used for both associativity methods. The second (logical) method is attempted first; if the logical associativity information is not located in the ExodusII file, the second method is used. The command line syntax for importing a mesh into CUBIT is:

Import Mesh 'exodusII_filename' Block <block_id> Volume <volume_id>

This command is also accessible in the GUI version as the **Import > Mesh** option under the **File** menu.

▼ Mesh Quality

The ‘quality’ of a mesh can be assessed using the element quality functions. These functions calculate several element shape factors which may have an affect on the accuracy of the finite element results calculated using the element.

Background

The quadrilateral element quality metrics that are calculated are aspect ratio, skew, taper, warpage, element area, and stretch. The calculations are based on an article by John Robinson entitled “CRE method of element testing and Jacobian shape parameters,” Eng. Comput., 1987, Vol. 4. An illustration of the shape parameters is shown in Figure 5-29 The warpage is calculated as the Z deviation from the ‘best-fit’ plane containing the element divided by the minimum of

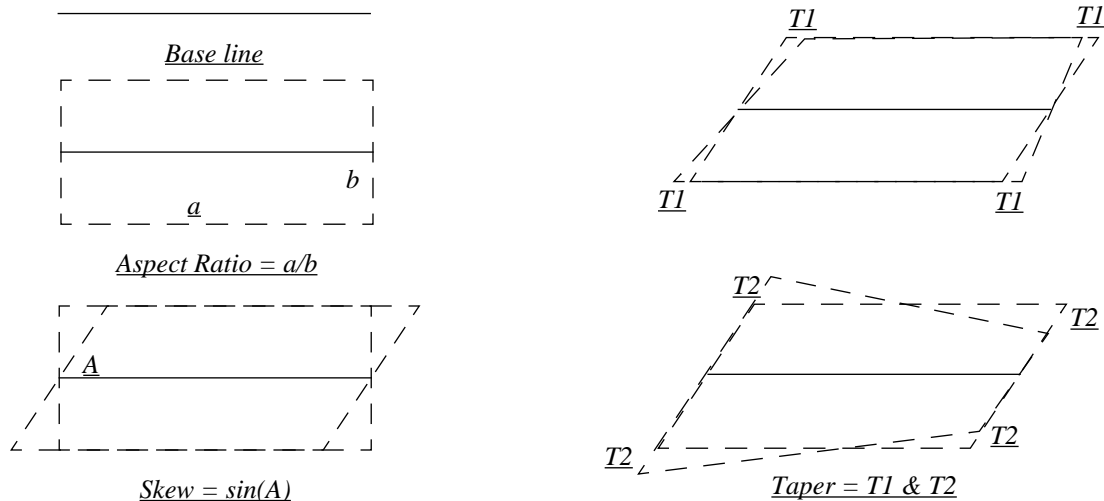


Figure 5-29 Illustration of Quadrilateral Shape Parameters (Quality Metrics)

‘a’ or ‘b’ from Figure 5-29. The stretch metric is calculated by dividing the length of the shortest element edge divided by the length of the longest element diagonal.

The hexahedral element quality metrics that are calculated are aspect ratio, skew, taper, element volume, stretch, and diagonal ratio. The calculation of these metrics is similar to that used for the quadrilateral elements. A good illustration and discussion of the mode shapes for an eight-node hexahedral element can be found in Chapter 3 of L. M. Taylor and D. P. Flanagan, “PRONTO 3D: A Three-Dimensional Transient Solid Dynamics Program,” SAND87-1912.

Command Syntax

The commands to access the quality metrics are:

quality <entity_list> [global]

quality <entity_list> [global] display/draw ‘metric_name’

The **global** identifier indicates that all specified entities are to be treated as a single entity instead of as several distinct entities. Valid values for the **metric_name** identifier are: **Aspect Ratio**, **Skew**, **Taper**, **Element Area** (Quad Only), **Element Volume** (Hex Only), **Warpage** (Quad Only), **Stretch**, and **Diagonal Ratio** (Hex Only). The example section below shows the typical output.

Command Examples

quality surf all global

-- lists quality summary for all surfaces in model. One summary

quality surf all

-- lists quality summary for all surfaces in model. One summary per entity

quality group 1

-- lists quality for the RefEntities in the group. Determines the highest common dimension (hex/quad).

quality surf 1 surf 2 surf 9

-- lists summary for surfaces 1, 2, and 9

quality surf all global draw 'Aspect Ratio'

-- Draws color-coded plot of the aspect ratios of the element faces.

Example Output

The typical summary output from the command **quality surface 1** is shown in Table 5-1. Figure 5-30a shows the histogram output corresponding to the above summary. The colored

Table 5-1 Sample Output for 'Quality' Command

Surface 1 Element Metrics:					
Function Name	Average	Std Dev	Minimum (id)	Maximum (id)	
Aspect Ratio	1.272e+00	2.336e-01	1.000e+00 (86)	2.200e+00 (433)	
Skew	2.035e-01	1.790e-01	7.168e-04 (121)	7.778e-01 (280)	
Taper	1.529e-01	1.048e-01	4.783e-03 (254)	6.842e-01 (70)	
Warpage	0.000e+00	0.000e+00	0.000e+00 (1)	0.000e+00 (1)	
Element Area	5.244e-04	5.683e-04	3.305e-05 (154)	2.371e-03 (229)	
Stretch	7.467e-01	1.136e-01	2.983e-01 (433)	9.648e-01 (310)	

element display resulting from the command **quality surface 1 draw 'Skew'** is shown in Figure 5-30b. In addition, a legend () is output to the terminal.

Table 5-1 Element Quality Plot Legend

Magenta ranges from 7.168e-04 to 1.117e-01 (127 entities)
Blue ranges from 1.117e-01 to 2.227e-01 (82 entities)
Cyan ranges from 2.227e-01 to 3.338e-01 (41 entities)
Green ranges from 3.338e-01 to 4.448e-01 (35 entities)
Yellow ranges from 4.448e-01 to 5.558e-01 (20 entities)
Orange ranges from 5.558e-01 to 6.668e-01 (11 entities)
Red ranges from 6.668e-01 to 7.778e-01 (8 entities)

Limitations and Planned Enhancements:

- The legend for the color plots is only a text output that tells the range covered by each color and the number of elements in that range.

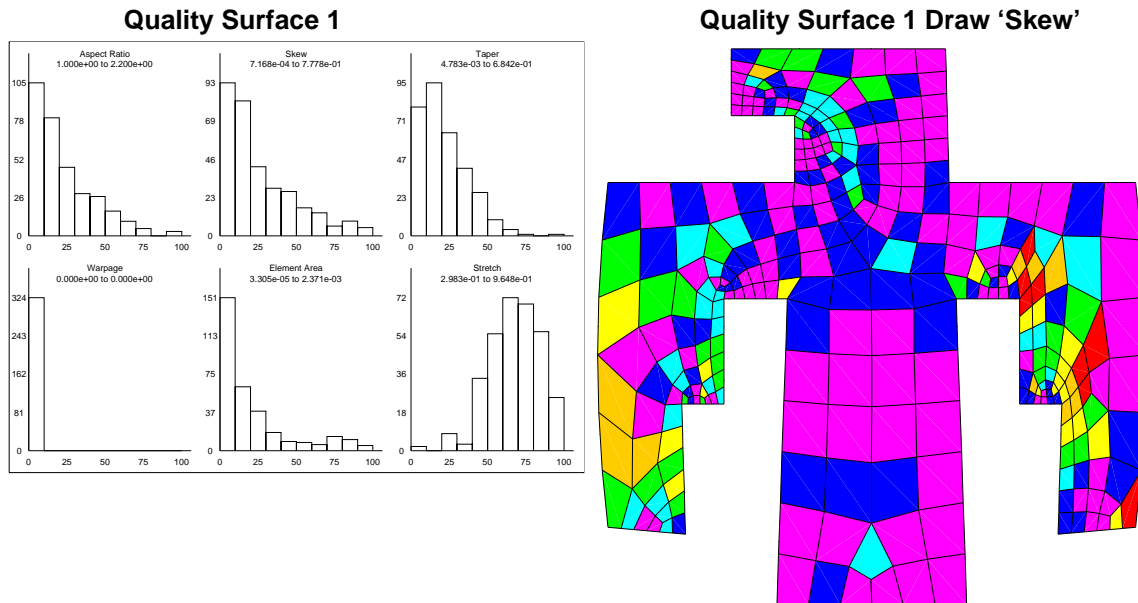
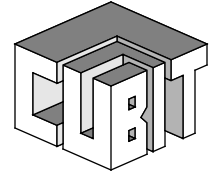


Figure 5-30 Illustration of Quality Metric Graphical Output

- Individual metrics can not currently be listed.
- The color quality plots change the graphics mode to polygon fill mode. The user must reset it back if a different mode is wanted.
- The histogram plot sets the background to white so that the hardcopy will work correctly. Need some way of resetting back.
- Recalculation of the metrics is done each time. Need some way of determining when the old data are valid/invalid. A function exists to set this, but there are too many places that the mesh data could be modified to rely on it being set correctly at this time. .



Chapter 6: Finite Element Model Definition and Output

- ▼ Finite Element Model Definition...123
 - ▼ Element Block Specification...124
- ▼ Boundary Conditions—Nodesets and Sidesets...126
 - ▼ Setting the Title...127
- ▼ Exporting the Finite Element Model...127

Chapter 6 describes the techniques used to complete the definition of the finite element model and the commands to export the finite element mesh to an Exodus database file. The definitions of the basic items in an Exodus database are briefly presented, followed by a description of the commands a user would typically enter to produce a customized finite element problem description.

▼ Finite Element Model Definition

Sandia's finite element analysis codes have been written to transfer mesh definition data in the ExodusII [6] file format. CUBIT is one code in a suite of computer codes that supports the ExodusII format for the pre- and post-processing of finite element analyses [14]. Since CUBIT is dedicated to mesh generation, the resulting database exported during a CUBIT session is sometimes referred to as a Genesis database file. A Genesis file is a subset of an Exodus file containing the problem definition only, i.e., no analysis results are included in the database.

A Genesis database consists of the following basic entity types: Element Blocks, Nodesets, and Sidesets.

Element Blocks

Element Blocks (also referred to as simply, *Blocks*) are a logical grouping of *elements* all having the same basic geometry and number of nodes. All elements within an Element Block are required to have the same element type. Access to an Element Block is accomplished through the use of a single integer ID known as the Block ID. Typically, Element Blocks are used by analysis codes to associate material properties and/or body forces with a group of elements.

Nodesets

Nodesets are a logical grouping of *nodes* also accessed through a single ID known as the Nodeset ID. Nodesets provide a means to specify load or boundary conditions on the CUBIT model.

Sidesets

Sidesets are another mechanism by which constraints may be applied to the model. Sidesets represent a grouping of *element sides* and are also referenced using an integer Sideset ID. They are typically used in situations where a constraint must be associated with element sides to satisfactorily represent the physics (e.g. a contact surface).

▼ Element Block Specification

Element blocks are the method CUBIT uses to group related sets of elements into a single entity. Each element in an element block must have the same dimensionality, type, number of nodes, and attributes. Element Blocks may be defined for volumes, surfaces, and curves. Multiple volumes, surfaces, and curves can be contained in a single element block. Element blocks are defined in the Block Identifier Dialog (Figure 6-1) which is accessed from the Block Identifier menu item. The **Block ID** and the **Geometry Type** to which this block ID is to be applied is specified in the top section of the dialog. The entities of this geometry type are specified using the normal picker window in the bottom section of the dialog. The center section of the dialog is used to specify the characteristics (element type and element attributes) to be applied to this element block. The element type desired is specified by selecting one of the radio buttons. The elements along the top row are basic linear elements and the subsequent rows are higher-order elements. The number following the element name denotes the number of nodes in the element. For example, the **Hex27** element is a 27-noded hexahedral element with mid-side, mid-face, and mid-volume nodes. The Shell and Bar elements require the specification of an **Attribute**¹ value which defines the thickness or cross-sectional area of the element for use in the finite element code². The attribute defaults to 1.0 if not specified. The commands to perform these functions using the command line are:

Block <block_id> {Curve | Surface | Volume} <range>

Block <block_range> Element Type <type>

Block <block_range> Attribute <value>

Where the first command defines a **block_id** containing the specified geometric entities, the second command sets the **Element Type** for that block and the third command sets the **Attribute** for those elements.

1. Only zero or one attributes can be defined at the current time. This limitation will be removed in a future version.
2. The thickness and cross-section attribute values are not used internally in CUBIT, they are merely flags which are written to the EXODUS file to be used by subsequent codes. The documentation for the code which will be reading the EXODUS file should be consulted to determine the correct specification and use of the attribute value for the Shell and Bar elements.



Note: Higher order element blocks **must** be specified prior to meshing since additional nodes are inserted as part of the meshing process *only* if an Element Block's element type calls for them.

Default Element Types, Block IDs, and Attributes

The following defaults will be used unless otherwise specified or modified:

Volume: The default block ID will be set to the Volume ID and 8-node hexahedral elements will be generated.

Surface: The block ID will be set to 0 and 4-node shell elements will be generated.

Curve: The block ID will be set to 0 and 2-node bar elements will be generated.

Meshing could then be accomplished and the desired finite element model exported to the Genesis database.

Element Block Definition Examples

Multiple Element Blocks

Multiple element blocks can and almost always are combined when generating a finite element mesh. For example if the finite element model consists of a block which has a thin shell encasing the volume mesh, the following block commands would be used:

```
Block 100 Volume 1
Block 100 Element Type Hex8
Block 200 Surface 1 To 6
Block 200 Element Type Shell14
Block 200 Attribute 0.01
Mesh Volume 1
Export Genesis 'block.g'
```

Which defines two element blocks (100 and 200). Element block 100 is composed of 8-node hexahedral elements and element block 200 is composed of 4-node shell elements on the surface of the block. The “thickness” of the shell elements is 0.01. The finite element code which reads the Genesis file (block.g) would refer to these blocks using the element block IDs 100 and 200. Note that the second line and the fourth line of the example are not required since both commands represent the default element type for the respective element blocks.

Surface Mesh Only

If a mesh containing only the surface of the block is desired, the first two lines of the example would be omitted and the **Mesh Volume 1** line would be changed to, for example, **Mesh Surface 1 To 6**.

Two-Dimensional Mesh

CUBIT also provides the capability of writing two-dimensional Genesis databases similar to FASTQ. The user *must* first assign the appropriate surfaces in the model to an element block. Then a **Quad*** type element may be specified for the element block. For example

```
Block 1 Surface 1 To 4
Block 1 Element Type Quad4
```

In this case, it is important for users to note that a two-dimensional Genesis database will result. In writing a two-dimensional Genesis database, CUBIT *ignores* all z-coordinate data.

Therefore, the user must ensure that the Element Block is assigned to a planar surface lying in a plane parallel to the x-y plane. Currently, the **Quad*** element types are the only supported two-dimensional elements. Two-dimensional shell elements will be added in the near future if required.

▼ Boundary Conditions—Nodesets and Sidesets

Boundary conditions such as constraints and loads are applied to the finite element model through nodesets and sidesets. Nodesets can be created from groups of nodes categorized by their owning volumes, surfaces, or curves. Nodes can belong to more than one nodeset. Sidesets can be created from groups of element sides or faces categorized by their owning surfaces or curves. Element sides and faces can belong to more than one sideset. Nodesets and Sidesets can be viewed individually through CUBIT by employing the **Draw Nodeset** and **Draw Sideset** commands.

Nodesets and Sidesets may be assigned to the appropriate geometric entities in the model using the following commands in the command line:

Nodeset <nodeset_id> {Curve | Surface | Volume | Vertex} <range>

Sideset <sideset_id> {Curve | Surface} <range>

When using the GUI version of CUBIT, nodesets and sidesets are specified by accessing their respective dialog boxes from the **Constraints** menu. The Nodesets menu item will display the Nodeset Dialog and the Sidesets menu item will display the Sideset Dialog. The top portion of both of these are shown in Figure 6-2; the bottom portion is a standard picker window. The Geometry Type option menu can be set to **Body**, **Volume**, **Surface**, or **Curve**. The user-specified output identification number for the nodeset or sideset is entered in the **Nodeset ID** or **Sideset ID** text field. This is the number which will be used to identify this boundary condition in the exported EXODUSII file. The geometric entities to which this boundary condition is to be applied is then specified using the normal picking syntax.

Nodeset Associativity Data

Nodesets are also used to store geometry associativity data in the ExodusII file. This data can be used to associate the corresponding mesh to an existing geometry in a subsequent CUBIT session. This functionality can be used either to associate a previously-generated mesh with a geometry (“Import Mesh” on page 119), or to associate a field function with a geometry for field function-based meshing “Sizing Function-Based Node Placement” on page 97).

The command syntax used to control whether or not associativity data is written to the ExodusII files is the following:

NodeSet Associativity { on | off }

▼ Setting the Title

CUBIT will automatically generate a default title for the Genesis database. The default title has the form:

```
cubit(genesis_filename): date: time
```

The title can be changed using the command:

```
Title '<title_string>'
```

CUBIT's parser requires that strings be enclosed in single quotes, for example **'This is a string'**.

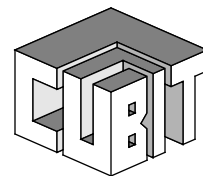
▼ Exporting the Finite Element Model

A Genesis database can be output using the **Export** option of the **File** Menu. Only Exodus II format is currently supported¹. A file can also be output using the following command:

```
Export Genesis '<filename>'
```

The Export Genesis command automatically creates a *unique* Element Block for every volume that is meshed (assuming the user has not entered any Block commands overriding this default behavior). Users can selectively control which blocks are output to the Genesis file since Element Blocks will *not* be created for any volumes that are not meshed.

¹. Actually, there are two other formats provided for specialized applications. These formats are Xpatch and FRED. The command to create these file formats is **Export Xpatch|FRED '<filename>'**. If you need more information about these file formats, contact a CUBIT developer.



Appendix A: Command Index

▼ Command Syntax...129

▼ Commands...129

In this section the commands available in CUBIT are listed in alphabetical order. There may be more than one command available under a given command definition, since some commands can be executed singly or over a range of objects.

▼ Command Syntax

Each command will be first listed by a command heading, which will be phrased in standard terminology and will typically be very close to the computer syntax for the particular command. Underneath the command heading, each variation of the command will be listed, to document commands which can be applied to single objects as well as a range of objects. At the end of each command heading is a page number cross-reference to the location in the main document where the command is documented.

Command listings will ask for four types of command arguments: arguments include integers (6, 1, 3, etc., these are typically id's), reals (1.4, 2.35, etc., typically floating point quantities such as angles, spatial coordinates), strings (jjrome.jou, part.g, etc.), and logicals (1, 0, on, off, etc.).

Detailed descriptions of most commands and their usage can be found within the main sections of this document. A few of the special purpose commands are not documented elsewhere and only their syntax is shown below. These commands will be fully-documented after the functionality and generality are improved.

▼ Commands

At	page 48
[View] At <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
Block Attribute	page 124
Block <block_id_range> Attribute <attribute>	
Block Element Type	page 124
Block <block_id_range> Element Type <element_type_name>	
Block Geometry Type	page 124
Block <id> {volume surface curve } <block_id_range>	

Block Label

Block <block_id_range> Label { on | off }

Block Visibility

page 52

Block <block_id_range> Visibility { on | off }

Body Copy

page 79

Body <body_id_range> Copy [Move <x> <y> <z>]

Body <body_id_range> Copy [Reflect {x|y|z}]

Body <body_id_range> Copy [Reflect <x> <y> <z>]

Body <body_id_range> Copy [Rotate <angle> About {x|y|z}]

Body <body_id_range> Copy [Rotate <angle> About <x> <y> <z>]

Body <body_id_range> Copy [Scale <scale>]

Body Geometry Visibility

page 53

Body <body_id_range> Geometry { on | off }

Body <body_id_range> Geometry Visibility { on | off }

Body Interval

page 97

Body <body_id_range> Interval <interval>

Body Label

Body <body_id_range> Label {on | off | name | id | interval}

Body Mesh Visibility

page 53

Body <body_id_range> Mesh { on | off }

Body <body_id_range> Mesh Visibility { on | off }

Body Move

page 79

Body <body_id_range> [Copy] Move <x> <y> <z>

Body Reflect

page 81

Body <body_id_range> [Copy] Reflect {x|y|z}

Body <body_id_range> [Copy] Reflect<x> <y> <z>

Body Restore

page 82

Body <body_id_range> Restore

Body Rotate

page 81

Body <body_id_range> [Copy] Rotate <angle> About {x|y|z}

Body <body_id_range> [Copy] Rotate <angle> About <x> <y> <z>

Body Scale

page 80

Body <body_id_range> [Copy] Scale <scale>

Body Size

page 97

Body <body_id_range> Size <size>

Body Visibility


page 53


Body <body_id_range> { on | off }

Body <body_id_range> Visibility { on | off }

	BoundaryLayer	page 107
	BoundaryLayer <layer_id_range> First [Layer] <depth> Growth [Factor] <growth> Total Depth <depth> [Sublayer <depth>] BoundaryLayer <layer_id_range> First [Layer] <depth> Growth [Factor] <growth> Layers <count> [Sublayer <depth>]	
	BoundaryLayer Surface	page 107
	BoundaryLayer <layer_id> Curve <curve_id_range> Surface <surface_id>	
	Brick	page 71
	[Create] Brick Width <width> [Depth <depth> Height <height>]	
	Check	
	Check {bodies surfaces curves}	
	Color Background	page 44
	Color Background <color_name> Color Background <color_id>	
	Color Block	page 53
	Color Block <block_id_range> <color_name>	
	Color Body	page 53
	Color Body <body_id_range> <color_name> Color Body <body_id_range> <color_id>	
	Color Body Geometry	page 53
	Color Body <body_id> Geometry <color_name> Color Body <body_id> Geometry <color_id>	
	Color Body Mesh	page 53
	Color Body <body_id_range> Mesh <color_name> Color Body <body_id_range> Mesh <color_id>	
	Color Group	
	Color Group <group_id_range> <color_name> Color Group <group_id_range> <color_id>	
	Color Group Geometry	
	Color Group <group_id_range> Geometry <color_name> Color Group <group_id_range> Geometry <color_id>	
	Color Node	page 53
	Color Node <color>	
	Color NodeSet	page 53
	Color NodeSet <nodeset_id_range> <color>	
	Color SideSet	page 53
	Color SideSet <sideset_id_range> <color>	

	Color Surface	page 53
	Color Surface <surface_id_range> <color>	
	Color Surface <surface_id_range> <color_id>	
	Color Surface Geometry	page 53
	Color Surface <surface_id_range> Geometry <color>	
	Color Surface <surface_id_range> Geometry <color_id>	
	Color Surface Mesh	page 53
	Color Surface <surface_id_range> Mesh <color>	
	Color Surface <surface_id_range> Mesh <color_id>	
	Color Volume	page 53
	Color Volume <volume_id_range> <color>	
	Color Volume <volume_id_range> <color_id>	
	Color Volume Geometry	page 53
	Color Volume <volume_id_range> Geometry <color>	
	Color Volume <volume_id_range> Geometry <color_id>	
	Color Volume Mesh	page 53
	Color Volume <volume_id_range> Mesh <color>	
	Color Volume <volume_id_range> Mesh <color_id>	
	Color Sheet	page 53
	Color Sheet <sheet_id_range> <color>	
	Color Sheet <sheet_id_range> <color_id>	
	Comment	
	Comment 'text written to journal file'	
	Copy Mesh	page 114
	Copy Mesh Surface <surface_id> Onto Surface <surface_id>	
	Copy Mesh Volume <volume_id> Onto Volume <volume_id>	
	Create Brick	page 71
	[Create] Brick Width <width> [Depth <depth> Height <height>]	
	Create Cylinder	page 71
	[Create] Cylinder Height <height> Radius <radius>	
	[Create] Cylinder Height <height> Major Radius <radius> Minor Radius <radius>	
	Create Frustum	page 73
	[Create] Frustum Height <height> Major Radius <radius>	
	Minor Radius <radius> [Top <top_radius>]	
	[Create] Frustum Height <height> Radius <radius> [Top <top_radius>]	
	Create Prism	page 72
	[Create] Prism Height <height> Sides <sides> Major <radius>	
	Minor <radius>	
	[Create] Prism Height <height> Sides <sides> Radius <radius>	

	Create Pyramid	page 74
	[Create] Pyramid Height <height> Sides <sides> Major <radius> Minor <radius> Top <radius> [Create] Pyramid Height <height> Sides <sides> Radius <radius>	
	Create Sphere	page 74
	[Create] Sphere Radius <radius> [Create] Sphere Radius <radius> [Inner Radius <inner_radius>] [xpositive] [ypositive] [zpositive] [delete]	
	Create Torus	page 75
	[Create] Torus Rad1 <R1> Rad2 <R2>	
	Curve Interval	page 97
	Curve <curve_id_range> Interval <interval>	
	Curve Label	
	Curve <curve_id_range> Label {on off name id interval}	
	Curve Reverse Bias	page 97
	Curve <curve_id_range> ReverseBias	
	Curve Scheme Curvature	
	Curve <curve_id_range> Scheme Curvature	
	Curve Scheme Bias	page 97
	Curve <curve_id_range> Scheme Bias Factor <growth_factor>	
	Curve Scheme Equal	page 97
	Curve <curve_id_range> Scheme Equal	
	Curve Size	page 97
	Curve <curve_id_range> Size <size>	
	Cylinder	page 71
	[Create] Cylinder Height <height> Radius <radius> [Create] Cylinder Height <height> Major Radius <radius> Minor Radius <radius>	
	Decompose	page 85
	Decompose <body_id> With <body_id>	
	Delete Body	
	Delete Body <body_id_range>	
	Delete Face	page 118
	Delete Face <face_id> Node1 <node1_id> Node2 <node2_id>	

	Delete Mesh	page 117
	Delete Mesh	
	Delete Mesh Vertex <vertex_id_range>	
	Delete Mesh Curve <curve_id_range>	
	Delete Mesh Surface <surface_id_range>	
	Delete Mesh Volume <volume_id_range>	
	Delete Mesh Body <body_id_range>	
	Delete Mesh Group <group_id_range>	
	Display	page 44
	Display	
	Draw Block	page 51
	Draw Block <block_id_range>	
	Draw Body	page 51
	Draw Body <body_id_range>	
	Draw Curve	page 51
	Draw Curve <curve_id_range>	
	Draw Edge	page 51
	Draw Edge <edge_id_range>	
	Draw Face	page 51
	Draw Face <face_id_range>	
	Draw Group	
	Draw Group <group_id_range>	
	Draw Hex	page 51
	Draw Hex <hex_id_range>	
	 Draw Loop	Undocumented
	Draw Loop <loop_id_range>	
	Draw Node	page 51
	Draw Node <node_id_range>	
	Draw NodeSet	page 51
	Draw NodeSet <nodeset_id_range>	
	Draw SideSet	page 51
	Draw SideSet <sideset_id_range>	
	Draw Surface	page 51
	Draw Surface <surface_id_range>	
	Draw Vertex	page 51
	Draw Vertex <vertex_id_range>	
	Draw Volume	page 51
	Draw Volume <volume_id_range>	

	Echo	page 41
	[Set] Echo {on off}	
	Exit	page 41
	Exit	
	Quit	
	Export	page 127
	Export Fred '<filename>'	
	Export Genesis '<filename>'	
	Export Xpatch '<filename>'	
	From	page 48
	[View] From <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
	Frustum	page 73
	[Create] Frustum Height <height> Major Radius <radius>	
	Minor Radius <radius> [Top <top_radius>]	
	[Create] Frustum Height <height> Radius <radius> [Top <top_radius>]	
	Geometry Visibility	page 52
	Geometry { on off }	
	Geometry Visibility { on off }	
	Graphics Autocenter	page 46
	Graphics Autocenter {on off}	
	Graphics Autoclear	page 46
	Graphics Autoclear {on off }	
	Graphics Axis	page 47
	Graphics Axis {on off}	
	Graphics Border	page 46
	Graphics Border {on off}	
	Graphics Center	page 47
	Graphics Center	
	Graphics Clear	page 47
	Graphics Clear	
	Graphics LineWidth	page 47
	Graphics LineWidth <width>	
	Graphics Mode	page 45
	Graphics Mode FlatShade	
	Graphics Mode HiddenLine	
	Graphics Mode PolygonFill	
	Graphics Mode Painters	
	Graphics Mode SmoothShade	
	Graphics Mode WireFrame	
	Graphics Mode Dual	

Graphics Pan

[Graphics] Pan {Left | Right | Up | Down} <factor> Animation Steps <number>
 [Graphics] Pan Cursor Animation Steps <number>

Graphics Perspective

page 49

Graphics Perspective {on | off}

Graphics Perspective Angle

page 49

Graphics Perspective Angle <view_angle_in_degrees>

Graphics Text Size

page 54

Graphics Text Size <size_factor>

Graphics Window

Graphics Window Active <window_number>
 Graphics Window Create <window_number>
 Graphics Window Delete <window_number>

Graphics WindowSize

page 44

Graphics WindowSize <width_in_pixels> <height_in_pixels>
 Graphics WindowSize Maximum

Graphics Zoom

page 50

[Graphics] Zoom <X_min> <Y_min> <X_max> <Y_max>
 Animation Steps <number>
 [Graphics] Zoom Cursor Animation Steps <number>
 [Graphics] Zoom Reset
 [Graphics] Zoom Screen <Scale_Factor> Animation Steps <number>

Group

Group 'group_name' Add <list_of_entity_ranges>
 Group <group_id> Add <list_of_entity_ranges>
 Group 'group_name' Remove <list_of_entity_ranges>
 Group <group_id> Remove <list_of_entity_ranges>

Group Interval

Group <group_id_range> Interval <interval>

Group Geometry Visibility

Group <group_id_range> Geometry Visibility { on | off }

Group Mesh Visibility

Group <group_id_range> Mesh Visibility { on | off }

Group Label

Group <group_id_range> Label { on | off | interval | id | name }





Group Size

Group <group_id_range> Size <size>

Hardcopy

page 55

Hardcopy '<filename>' [encapsulated|postscript|eps] [color|monochrome]
 Hardcopy '<filename>' Pict [XSize <xsize>] [YSize <ysize>]

	Help	page 65
	Help	
	Help <keyword>	
	<keyword> Help	
	 Hyperhelp	page 65
	Hyperhelp <keyword>	
	<keyword> [<identifier>] Hyperhelp	
	Import Acis	page 77
	Import Acis '<acis_filename>'	
	 Import Fastq	page 78
	Import Fastq '<fastq_filename>'	
	 Import Mesh	page 119
	Import Mesh <exodusII_filename> Block <block_id>	
	Volume <volume_id>	
	 Import Sizing Function	page 120
	Import Sizing Function '<exodusII_filename>' Block <block_id>	
	Variable '<variable_name>' Time <time_val>	
	Intersect	page 82
	Intersect <body_id> With <body_id>	
	Journal	page 42
	[Set] Journal {on off}	
	Label	page 54
	Label {on off interval id name }	
	Label All { on off interval id name }	
	Label Body { on off interval id name }	
	Label Curve { on off interval id name }	
	Label Edge { on off interval id name }	
	Label Face { on off }	
	Label Geometry { on off interval id name }	
	Label Group { on off interval id name }	
	Label Hex { on off }	
	Label Mesh { on off }	
	Label Node { on off }	
	Label Surface { on off interval id name }	
	Label Vertex { on off interval id name }	
	Label Volume { on off interval id name }	

List (Geometry/Mesh Related)

page 76

List Body <body_id_range> [{geometry|debug}]
 List Curve <curve_id_range> [{geometry|debug}]
 List Face <mesh_face_id_range>
 List Group <group_id_range> [{geometry|debug}]
 List Hex <hex_id_range>
 List Names [{Group|Body|Volume|Surface|Curve|Vertex}]
 List Node <node_id_range>
 List Surface <surface_id_range> [{geometry|debug}]
 List Totals
 List Model
 List Vertex <vertex_id_range> [{geometry|debug}]
 List Volume <volume_id_range> [{geometry|debug}]

List (Other)

page 62

List Debug
 List Echo
 List Information
 List Journal
 List Logging
 List Memory `<ClassName>`
 List Memory
 List Settings
 List View
 List Warning

Merge

page 87

Merge All
 Merge All Curves
 Merge All Surfaces
 Merge Body <body_id> With Body <body_id>
 Merge Body <body_id_range>
 Merge Curve <curve_id> With <curve_id>
 Merge Curve <curve_id_range>
 Merge Surface <surface_id> With <surface_id>
 Merge Surface <surface_id_range>

Mesh Body

page 114

Mesh Body <body_id_range>
 Mesh Body All

Mesh Curve

page 98

Mesh Curve <curve_id_range>
 Mesh Curve All


Mesh Group

Mesh Group <group_id_range>
 Mesh Group All

	Mesh Surface	page 107
	Mesh Surface <surface_id_range>	
	Mesh Surface All	
	Mesh Visibility	page 52
	Mesh { on off }	
	Mesh Visibility { on off }	
	Mesh Volume	page 114
	Mesh Volume <volume_id_range>	
	Mesh Volume All	
	Name	
	Name {Group Body Volume Surface Curve Vertex} <id> `entity_name`	
	Node Move	
	Node <node_id_range> Move <delta_x> <delta_y> <delta_z>	
	Node Visibility	page 52
	Node { on off }	
	Node Visibility { on off }	
	NodeSet Associativity	page 126
	NodeSet Associativity { on off }	
	NodeSet Curve	page 126
	NodeSet <nodeset_id> Curve <curve_id_range>	
	NodeSet Label	
	NodeSet <nodeset_id_range> Label {on off}	
	NodeSet Move	page 117
	NodeSet <nodeset_id> Move <X> <Y> <Z>	
	NodeSet <nodeset_id> Move To <X> <Y> <Z>	
	NodeSet Surface	page 126
	NodeSet <nodeset_id> Surface <surface_id_range>	
	NodeSet Vertex	page 126
	NodeSet <nodeset_id> Vertex <vertex_id_range>	
	NodeSet Visibility	page 52
	NodeSet { on off }	
	NodeSet Visibility { on off }	
	NodeSet <nodeset_id_range> { on off }	
	NodeSet <nodeset_id_range> Visibility { on off }	
	NodeSet Volume	page 126
	NodeSet <nodeset_id> Volume <volume_id_range>	
	Pan	
	[Graphics] Pan {Left Right Up Down} <factor> Animation Steps <number>	
	[Graphics] Pan Cursor Animation Steps <number>	

Pause	page 43
Pause	
Pick	
Pick {Curve Surface Volume Body [List]}	
Playback	page 43
Playback '<journal_filename> '	
Plot	
Plot	
Prism	page 72
[Create] Prism Height <height> Sides <sides> Major <radius> Minor <radius>	
[Create] Prism Height <height> Sides <sides> Radius <radius>	
Pyramid	page 74
[Create] Pyramid Height <height> Sides <sides> Major <radius> Minor <radius> Top <radius>	
[Create] Pyramid Height <height> Sides <sides> Radius <radius>	
Quality	
Quality <entity_list> [Global]	
Quality <entity_list> [Global] Display Draw 'Metric Name'	
Quit	page 41
Quit	
Exit	
Record	page 42
Record '<journal_filename>'	
Record Stop	page 43
Record Stop	
Reset	page 41
Reset	
Reset Blocks	
Reset Genesis	
Reset Nodesets	
Reset SideSets	
Rotate	page 49
Rotate <degree> About [Screen World Camera] {x y z} Animation Steps <number>	
Rotate <degree> About Curve <curve_id> Animation Steps <number>	
Rotate <degree> About Vertex <axis_start_vertex_id> Vertex <axis_end_vertex_id> Animation Steps <number>	

Set	page 63
[Set] Debug <flag_id> {on off} [{File 'filename' Terminal}] [Set] Echo {on off} [Set] Info {on off} [Set] Journal {on off} [Set] Logging {on off} [{File 'filename' Terminal}] [Set] Warning {on off}	
Sheet Visibility	page 52
Sheet <sheet_id_range> Visibility { on off }	
SideSet Curve	page 126
SideSet <sideset_id> Curve <curve_id_range>	
SideSet Label	
SideSet <sideset_id_range> Label { on off }	
SideSet Surface	page 126
SideSet <sideset_id> Surface <surface_id_range>	
SideSet Visibility	page 52
SideSet { on off } SideSet Visibility { on off } SideSet <sideset_id_range> { on off } SideSet <sideset_id_range> Visibility { on off }	
Smooth Group	
Smooth Group <group_id_range>	
Smooth Surface	page 115
Smooth Surface <surface_id_range> [Global]	
Smooth Volume	page 116
Smooth Volume <volume_id_range>	
Sphere	page 74
[Create] Sphere Radius <radius> [Create] Sphere Radius <radius> [Inner Radius <inner_radius>] [xpositive] [ypositive] [zpositive] [delete]	
Subtract	page 83
Subtract <body_id> From <body_id>	
Surface Angle	
Surface <surface_id_range> Angle <angle_degrees>	
Surface Geometry Visibility	page 53
Surface <surface_id_range> Geometry Visibility { on off }	
Surface Interval	page 97
Surface <surface_id_range> Interval <intervals>	

	Surface Label	
	Surface <surface_id_range> Label {on off name id interval}	
	Surface Mesh Visibility	page 53
	Surface <surface_id_range> Mesh Visibility { on off }	
	Surface Periodic Interval	
	Surface <surface_id_range> Periodiic Interval <intervals>	
	Surface Scheme Curvature	page 99
	Surface <surface_id_range> Scheme Curvature	
	Surface Scheme Map	page 99
	Surface <surface_id_range> Scheme Map	
	Surface Scheme Pave	page 99
	Surface <surface_id_range> Scheme Pave	
	Surface Scheme Triangle	page 99
	Surface <surface_id_range> Scheme Triangle	
	Surface Scheme TriMap	
	Surface <surface_id_range> Scheme TriMap	
	Surface Scheme TriPave	
	Surface <surface_id_range> Scheme TriPave	
	Surface Size	page 97
	Surface <surface_id_range> Size <intervals>	
	Surface Sizing Function	page 119, 120
	Surface < id > Sizing Function Type { Curvature Linear Interval Inverse Test Exodus} [Min <min_val> Max <max_val>]	
	Surface Smooth Scheme	page 115
	Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]	
	Surface <surface_id_range> Smooth Scheme Equipotential [Fixed] Weight Jacobian	
	Surface <surface_id_range> Smooth Scheme Equipotential [Fixed] Weight Area	
	Surface <surface_id_range> Smooth Scheme Equipotential [Fixed] Weight Inverse [Area]	
	Surface <surface_id_range> Smooth Scheme Laplacian [Fixed]	
	Surface <surface_id_range> Smooth Scheme Centroid Area Pull [Fixed]	
	Surface Visibility	page 52
	Surface <surface_id_range> { on off }	
	Surface <surface_id_range> Visibility { on off }	
	Title	page 127
	Title '<title>'	

Torus	page 75
[Create] Torus Major [Radius] <R1> Minor [Radius] <R2>	
Unite	page 83
Unite <body_id> With <body_id>	
Unite Body <body_id_list> [All]	
Up	page 48
[View] Up <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
Version	page 41
Version	
Vertex	
Vertex <vertex_id_range> {on off}	
Vertex Label	
Vertex <vertex_id_range> Label {on off name id interval}	
Vertex Visibility	page 52
Vertex { on off }	
Vertex Visibility { on off }	
Video	
Video { on off }	
Video Initialize	page 55
Video Initialize [<number_of_frames>]	
Video Initialize 'base_filename' pict [Xsize <xsize>] [Ysize <ysize>]	
Video Snap	page 56
Video Snap	
View At	page 48
[View] At <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
View From	page 48
[View] From <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
View List	page 50
View List	
View Reset	
View Reset	
View Up	page 48
[View] Up <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
Volume Geometry Visibility	page 53
Volume <volume_id_range> Geom { on off }	
Volume <volume_id_range> Geometry Visibility { on off }	
Volume Interval	page 97
Volume <volume_id_range> Interval <intervals>	

	Volume Label	
	Volume <volume_id_range> Label {on off name id interval}	
	Volume Mesh Visibility	page 53
	Volume <volume_id_range> Mesh { on off }	
	Volume <volume_id_range> Mesh Visibility { on off }	
	Volume Scheme Curvature	
	Volume <volume_id_range> Scheme Curvature	
	Volume Scheme Map	page 108
	Volume <volume_id_range> Scheme Map	
	Volume Scheme Plaster	page 108
	Volume <volume_id_range> Scheme Plaster	
	Volume Scheme Project	page 108
	Volume <volume_id> Scheme Project Source Surface <surface_id_list>	
	Target Surface <surface_id>	
	Volume Scheme Rotate	page 108
	Volume <volume_id> Scheme Rotate Source Surface <surface_id_list>	
	Target Surface <surface_id>	
	Volume Scheme Translate	page 108
	Volume <volume_id> Scheme Translate	
	Source Surface <surface_id_list> Target Surface <surface_id>	
	Volume Scheme Weave	page 108
	Volume <volume_id_range> Scheme Weave	
	Volume Size	page 97
	Volume <volume_id_range> Size <intervals>	
	Volume Smooth Scheme	page 115
	Volume <volume_id_range> Smooth Scheme Laplacian	
	Volume <volume_id_range> Smooth Scheme Equipotential [Fixed]	
	Volume Visibility	page 53
	Volume <volume_id_range> { on off }	
	Volume <volume_id_range> Visibility { on off }	
	WebCut Body	page 85
	Webcut Body <body_id> Face <face_id> [Vector <from_vertex> <to_vertex>]	
	Webcut Body <body_id> Vertex <vertex_1> Vertex <vertex_2>	
	Vertex <vertex_3> [Vector <from_vertex> <to_vertex>]	
	Weight Hexes	page 116
	Weight Hexes Surface <range> <weight>	

Zoom

page 50

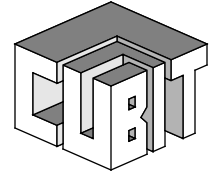
[Graphics] Zoom <X_min> <Y_min> <X_max> <Y_max>

Animation Steps <number>

[Graphics] Zoom Cursor Animation Step <number>

[Graphics] Zoom Screen <Scale_Factor> Animation Steps <number>

[Graphics] Zoom Reset



Appendix B: Examples

- ▼ General Comments...147
- ▼ Simple Internal Geometry Generation...148
 - ▼ Octant of Sphere...149
 - ▼ Airfoil...151
 - ▼ The Box Beam...152
- ▼ Thunderbird 3D Shell...155
- ▼ Assembly Components...158

The purpose of this Appendix is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a few examples on the use of CUBIT. Some examples also demonstrate the use of the ACIS test harness as well as other related programs. This Appendix is not intended to be a step-by-step tutorial.

▼ General Comments

CUBIT is based upon the ACIS solid modeling kernel. Solid models can be created within CUBIT or imported in the form of an ACIS geometry file¹. Current means of generating ACIS solid models external to CUBIT include:

- ACIS Test Harness
- FASTQ via the FASTQ to ACIS translator *fsqacs*²
- Aries[®] ConceptStation
- PRO/Engineer via a PRO/Engineer to ACIS translator

These examples show model construction using internal CUBIT geometry creation, the ACIS Test Harness, and the FASTQ translator. Those methods provide the capability of semi-automatically generating a mesh in batch mode in much the same manner as FASTQ [5], GEN3D [10], GREPOS [11], and GJOIN [12].

A CUBIT journal file is included for the examples shown in this appendix. ACIS journal files are also provided for the examples that require geometry generated by ACIS. The user can

1. ACIS typically adds the filename suffix “.sat” to the output files it writes in text format; therefore, these files are typically referred to as “.sat” files or “ACIS .sat” files. CUBIT cannot read binary ACIS files.
2. The *fsqacs* users manual is reproduced in Appendix C, “Fsqacs: A FASTQ to ACIS Command Interpreter” on page 163

reproduce the examples interactively by simply entering each of the lines in the journal files as commands to CUBIT or ACIS. The examples assume that the command line version of CUBIT will be used. The examples can also be run using the Graphical User Interface version of CUBIT; however, the details for doing this are not given in this appendix. The journal files included in the example are also distributed with CUBIT and they may be executed using the **Playback 'filename'** command.

The examples in this appendix each cover several of CUBIT's mesh generation capabilities. The CUBIT features exercised by each example are shown in Table B-1.

Examples	Geometry Features				Surface Meshing Features						Volume Meshing Features			
	Primitives	Booleans	Geometry Editing	Geometry Consolidation	Curve Bias	Curvature-based	Mapping	Paving	Triangle Tool	Boundary Layer Tool	Project	Translate/Rotate	Mapping	Plastering
Internal Geometry	x	x						x				x		
Sphere Octant	x	x	x	x				x	x		x	x		
Airfoil					x			x		x				
Box Beam				x			x							
Thunderbird								x						
Assembly Components				x				x			x			

Table B-1 CUBIT Features Exercised by Examples.

▼ Simple Internal Geometry Generation

This simple example demonstrates the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. It illustrates the **brick**, **cylinder**, **subtract**, **pave**, and **translate** commands and boolean operations. The geometry to be generated is shown in Figure B-1. This figure also shows the curve and surface labels specified in the CUBIT journal file. The final meshed body is shown in Figure B-2. The CUBIT journal file is:

Internal Geometry Generation Example

```
Brick Width 10. Depth 10. Height 10. # Create Cube
Cylinder Height 12. Radius 3. # Create cylinder through Cube
View From 3 4 5 # Update viewing position
Display
```

```

Subtract 2 From 1      # Remove cylinder from cube—create hole
Display
Body 3 Size 1.0        # Default element size for model
Surface 10 Interval 10  # Change intervals on cylinder surface
Curve 15 to 16 Interval 20 # Change intervals around cyl. circ.
Surface 11 Scheme Pave      # Front surface paved
Volume 3 Scheme Translate Source 11 Target 12      #Remainder
                                     # of block will be meshed by
                                     # translating front surface to back surface
Mesh Volume 3          # Create the mesh
Graphics Mode Hiddenline
Display                # Hiddenline view of cube (Figure B-2)

```

The first two lines create a 10 unit cube centered at the origin and a cylinder with radius 3 units and height of 12 units also centered at the origin. The cylinder height is arbitrary as long as it is greater than the height of the brick. The **subtract** command then performs the boolean by subtracting the cylinder (body 2) from the block (body 1) to create the final geometry (body 3). The remainder of the commands simply assign the desired number of intervals and then generate the mesh. Note that since the cylindrical hole is a “periodic surface,” there are no edges joining the two curves so the number of intervals along its axis must be set by the surface interval command. The steps required for generating this geometry and mesh using the Graphical User Interface are given in the Tutorial in Chapter 2.

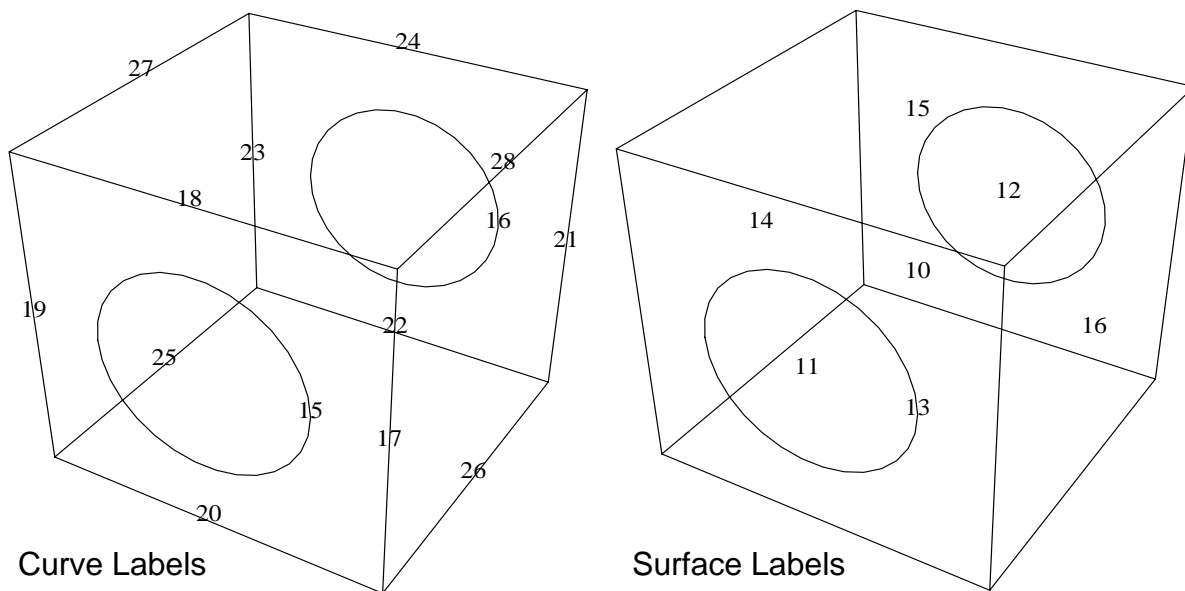


Figure B-1 Geometry for Cube with Cylindrical Hole

▼ Octant of Sphere

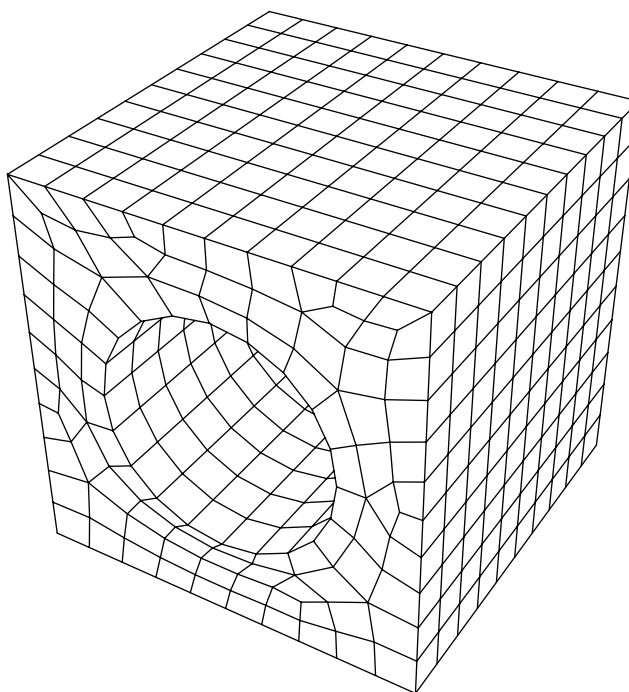
This example also illustrates the internal geometry generation capabilities of CUBIT to generate an octant of a sphere. The procedure used is to generate the octant by intersecting a brick with a sphere. The octant is then split into two pieces—a central “core” and an outer “peel” which are both meshable using the sweeping schemes. This example uses the **sphere**, **brick**, **cylinder**, **intersect**, **copy**, **subtract**, **merge**, **pave**, **project**, and **rotate** commands.

The following annotated CUBIT journal file will generate the mesh shown in Figure B-3.

```

Sphere Radius 10.          #Generate Sphere (Body 1)
Brick Width 12 Depth 12 Height 12  #Generate Cube (Body 2)

```

**Figure B-2** Generated Mesh for Cube with Cylindrical Hole

```

Body 2 Move 6. 6. 6.                                #Move Cube to Enclose Octant
Graphics Mode SmoothShade                            #Only way to see a sphere
Display
Intersect 1 With 2                                    #Generate Octant (Body 3)
Display
Cylinder Height 22 Radius 3                          #Generate Cylinder (Body 4)
Body 3 To 5 Copy                                     #Copy Octant (Body 5)
                                                    #and Cylinder (Body 6)
                                                    #and another octant (Body 7)
                                                    #Create Core (Body 8)

Intersect 4 With 5
View From 1 2 3
Intersect 3 With 6                                    #Create Another Core (Body 9)
Subtract 8 From 7                                     #Create Peel (Body 10)
Merge All                                             #Coalesce Redundant Surfaces
#
# End of Geometry Generation.
# "Core" is volume/body 9
# "Shell" is volume/body 10
#
volume 9 Size 0.5
Surface 33 Scheme Pave                                #Pave end of core
Mesh Surface 33
volume 9 Scheme Project Source 33 Target 31 #Generate core mesh
Mesh volume 9
Display                                              #Make sure it's there
#
volume 10 Size 0.5                                    #Make intervals agree for rotate
Surface 37 Scheme Pave                                #Pave face of peel
Mesh Surface 37
volume 10 Scheme Rotate Source 37 Target 40 #Generate Peel Mesh
Mesh volume 10
Display
Export Genesis 'Octant.gen'                          #Write out the mesh

```

If the generated mesh should consist of one material, the **block** command could be used to merge the peel and core into a single material block. Note that during a boolean operation (unite, intersect, and subtract), the bodies used in that boolean are destroyed so it is sometimes necessary to create extra copies of a body prior to using them in a boolean operation. Also,

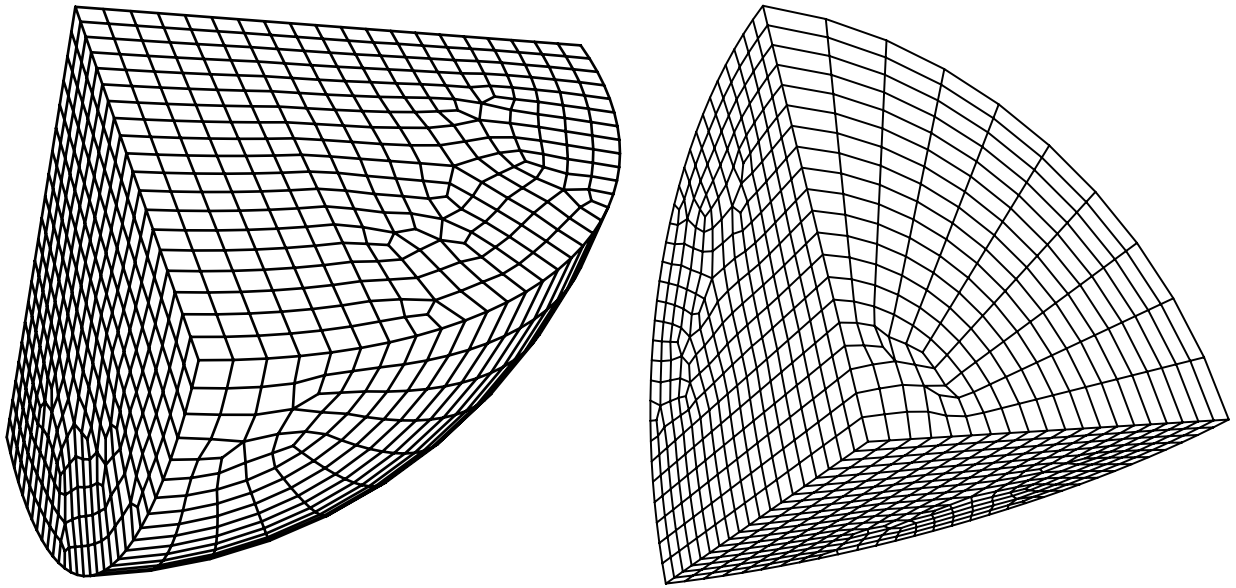


Figure B-3 Generated Mesh for Octant of Sphere

during boolean operations, many bodies are created and deleted and it is difficult to remember which bodies exist at certain times¹. It is recommended that comments be added to the journal file to make it easier to determine what is being done in the file.

▼ Airfoil

A simple two-dimensional airfoil is used in this example to demonstrate the use of the boundary layer tool and paving. The commands used to generate the geometry for this problem, using the ACIS Test Harness, are not included here. This example uses the **curve bias**, **boundarylayer** and **pave** commands. The CUBIT commands used to mesh this problem are:

```
# File: foil.jou
#
# Air Foil Example
#
journal off
Import Acis 'foil.sat'
View From 100 0 0 Up 0 0 1 # Set up View
AutoCenter On
Display
Volume 1 Interval 14 # Set Meshing Parameters
Curve 4 Interval 24
Curve 2 Interval 24
Curve 5 To 6 Interval 18
Curve 6 Bias 1.1
Curve 5 Bias 0.909
BoundaryLayer 1 First Layer 0.5 Growth 1.3
BoundaryLayer 1 Surface 1 Curve 5 to 6
Surface 1 Scheme Pave
Mesh Surface 1 # Create the Mesh
Display
```

1. The CUBIT Developers are very much aware of the problems this causes during the generation of complicated meshes and are implementing methods to permit user-defined naming of bodies and volumes. This capability relies on the persistent ID concept recently added to ACIS.

```
Graphics zoom .25 .4 .45 .6
geometry visibility off
display
geometry visibility on
```

The mesh generated by these commands is shown in Figure B-4. In this example, curves 5 and 6 (the curves used to define the shape of the airfoil) use biased interval spacing to place more elements towards the front of the airfoil. A boundary layer is designated on either side of the airfoil, which produces elements with high aspect ratios for several layers around the airfoil. The parameters to the **boundarylayer** command specify the depth of the first and second rows of elements, with the boundary layer growth factor inferred from these data. The paving scheme generates the mesh outside the boundary layer.

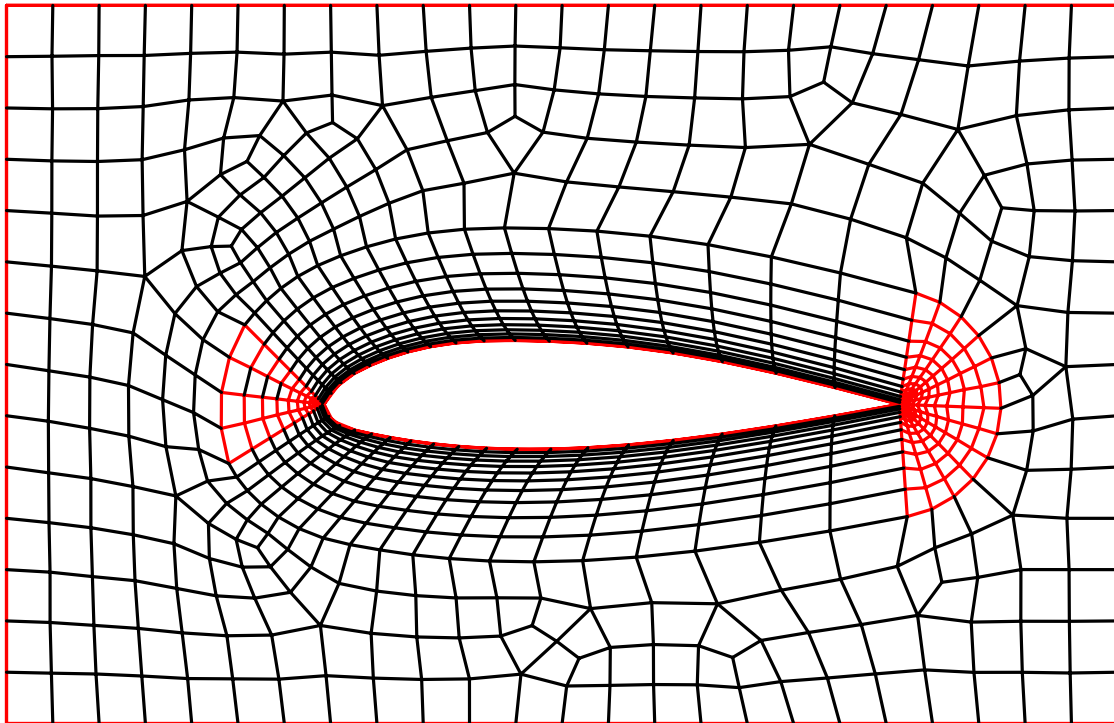


Figure B-4 Airfoil mesh generated using the boundary layer tool and paving.

▼ The Box Beam

A simple example using ACIS/CUBIT is the box beam buckling problem shown in Figure B-5. A description of an analysis which uses this type of mesh is found in Reference [15]. This example uses the **merge**, **nodeset** and **block** commands and the mapping mesh generation scheme.

The input file for the ACIS Test Harness for the box beam example is¹:

1. This file must be preprocessed by Aprepro prior to being input to the ACIS Test Harness.

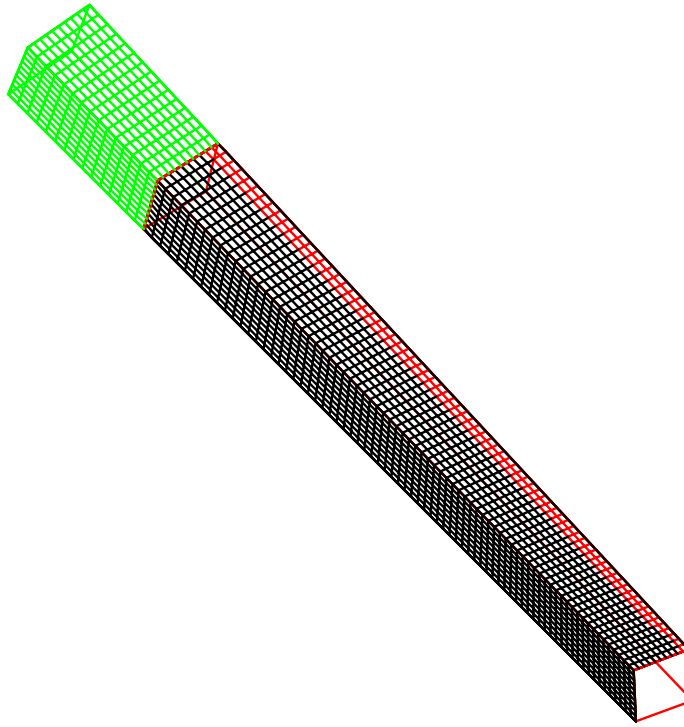


Figure B-5 Box Beam example

```
# File: boxBeam.mon
# Side = {Side = 1.75}
# Height = {Height = 12.0}
# Upper = {Upper = 2.0}

block lowerSection width {Side/2.0} depth {Side/2.0} height
{Height - Upper}
block upperSection width {Side/2.0} depth {Side/2.0} height
{Upper}

move lowerSection {Side/4.0} {Side/4.0} {(Height - Upper)/2.0}
move upperSection {Side/4.0} {Side/4.0} {Upper/2.0 + Height -
Upper}

group lowerSection upperSection as boxBeam

save boxBeam to boxBeam.sat
```

In this example, it is assumed that subsequent analyses will take advantage of the problem symmetry and therefore only one-quarter of the box beam will be meshed. It is worth noting that there are a variety of ways to construct a solid model for this problem; however, experience thus far with ACIS and CUBIT indicates that the easiest way to model the box beam is to use ACIS block primitives¹. Even though subsequent meshing will only be performed on the faces of the solid model, the entire 3D body is saved as an ACIS.sat file. The CUBIT journal file for the box beam example is:

1. This geometry can also be generated using the internal CUBIT Brick primitive.

File: boxBeam.jou

```

# Thickness = {Thickness = 0.06}
# Crease = {Crease = 0.01}
# XYInts = {XYInts = 10}
# ZInts = {ZInts = 90}
# UpperInts = {UpperInts = 15}

Import Acis 'boxBeam.sat'
#Display

Merge All
Label Surface
#Display

Label Curve
#Display

Curve 1 To 8 Interval {XYInts}
Curve 13 To 16 Interval {XYInts}

Curve 9 To 12 Interval {ZInts-UpperInts}
Curve 21 To 24 Interval {UpperInts}

Mesh Surface 3
Mesh Surface 6
Mesh Surface 9
Mesh Surface 12

NodeSet 1 Curve 1
NodeSet 2 Curve 4

NodeSet 1 Move {-Crease} 0 0
NodeSet 2 Move 0 {Crease} 0

Block 2 Surface 3
Block 2 Surface 6

Block 1 Surface 9
Block 1 Surface 12

Block 1 To 2 Attribute {Thickness}

Export Genesis 'boxBeam.exoII'
Quit

```

Commands worth noting in the CUBIT journal file include:

- **Block, Block Attribute** Allows the user to specify that shell elements for the surfaces of the solid model are to be written to the output (EXODUSII) database, and that shell elements be given a thickness attribute. This is necessary since CUBIT defaults to three-dimensional hexahedral meshing of solid model volumes.
- **NodeSet Move** Allows the user to actually move the specified nodes by a vector (Δx , Δy , Δz). This is advantageous for the buckling problem, since the numerical simulation requires a small “crease” in the beam in order to perform well.
- **Merge** Allows the user to combine geometric features (e.g. edges and surfaces).

Other commands in the journal file should be straightforward. Since the problem is sufficiently simple to mesh using a mapping transformation, specification of a meshing “scheme” is unnecessary (mapping is the default in CUBIT).

Finally, note that both the ACIS monitor file (**boxBeam.mon**) and the CUBIT journal file (**boxBeam.jou**) contain macros that are evaluated using Aprepro. The *makefile* used to semi-automatically generate the mesh is given below:

File: Makefile

```
boxBeam.g:boxBeam.exoII
    exo2exol boxBeam.exoII boxBeam.g

boxBeam.exoII:boxBeam.sat boxBeam.jou
    aprepro boxBeam.jou | cubitb
    rm cubit.jou

boxBeam.sat: boxBeam.mon
    aprepro boxBeam.mon | acis
    rm wjbohnhl.*

clean:
    @-rm *.sat *.exoII *.g
```

While this particular example is a trivial use of the software, it does serve to demonstrate a few of the capabilities offered by ACIS and CUBIT.

▼ Thunderbird 3D Shell

This example is the three-dimensional paving of a shell shown in Figure B-6. The 2D wireframe geometry of the thunderbird is given by the following FASTQ file:

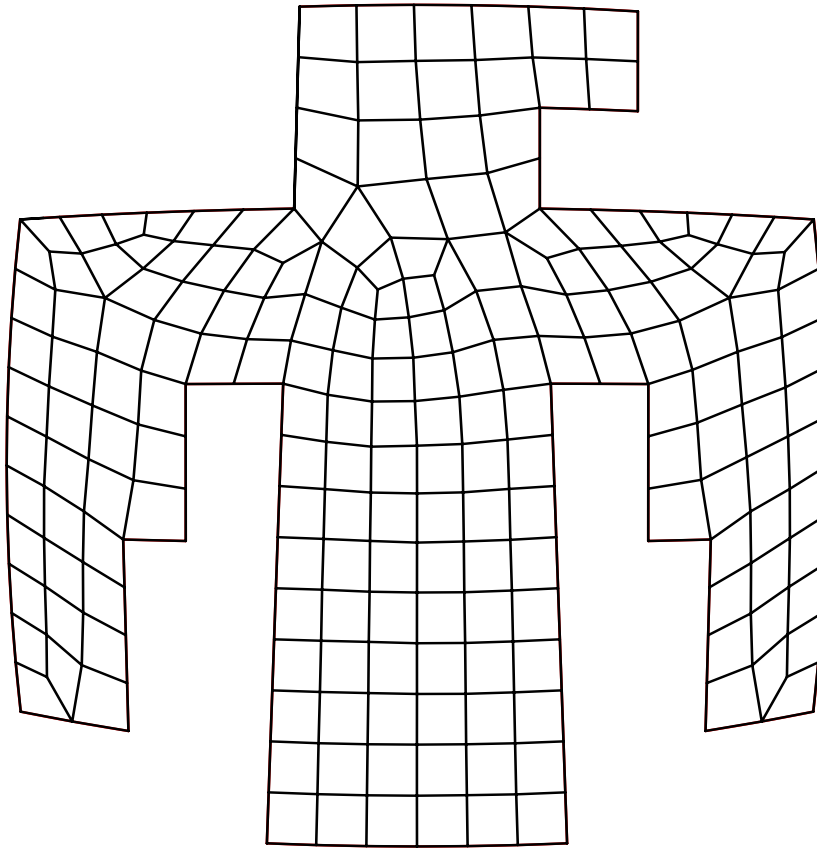


Figure B-6 Sandia Thunderbird 3D shell

#File: tbird.fsq

TITLE

MESH OF SANDIA THUNDERBIRD

```

$ block {e = .2} int= {isq = 20}
$ number of elements in block thick {iblkt = 5 } block thickness
{blkt=.2 }
$ block angle {angle=15}
$ magnification factor = {magnificationFactor=1.0}
$ bird {bthick = .018} {ithick = 3} {idepth = 20}
$ {pi = 3.14159265359} {rad=magnificationFactor/pi} {bdepth=1.}
$ preferred normalized element size = {elementSize=0.06}
$ number of intervals along outside edges =
$ {border_int=5} {corner_int=10} {side_int=20}
$ {outsideIntervals= 2*corner_int+side_int}
$ {boxTop=.2} {topIntervals = 8}

```

```

$ {insideCurveInt=8}

```

```

$ {MAG=magnificationFactor/3.0}

```

```

$ {middleInside=MAG*0.97}
$ {xCurveStartInside=MAG*0.60}
$ {yCurveStartInside=MAG*0.93}
$ {curveMiddleInside=MAG*0.81}

```

```

$ {xCurveStartOutside=MAG*0.75}
$ {yCurveStartOutside=MAG*1.17}
$ {middleOutside=MAG*1.20}
$ {curveMiddleOutside=MAG*1.01}
$ {boundingBox = MAG*1.5}

```

\$ Thunderbird Coordinates

```

POINT 1 {MAG*-.40} {MAG*.78}
POINT 2 {MAG*-.40} {MAG*.59}
POINT 3 {MAG*-.22} {MAG*.59}
POINT 4 {MAG*-.22} {MAG*.40}
POINT 5 {MAG*-.75} {MAG*.40}
POINT 6 {MAG*-.78} {MAG*-.09}
POINT 7 {MAG*-.75} {MAG*-.58}
POINT 8 {MAG*-.53} {MAG*-.60}
POINT 9 {MAG*-.54} {MAG*-.23}
POINT 10 {MAG*-.42} {MAG*-.23}
POINT 11 {MAG*-.42} {MAG*.07}
POINT 12 {MAG*-.24} {MAG*.07}
POINT 13 {MAG*-.27} {MAG*-.80}
POINT 14 {MAG*.27} {MAG*-.80}
POINT 15 {MAG*.24} {MAG*.07}
POINT 16 {MAG*.42} {MAG*.07}
POINT 17 {MAG*.42} {MAG*-.23}
POINT 18 {MAG*.54} {MAG*-.23}
POINT 19 {MAG*.53} {MAG*-.60}
POINT 20 {MAG*.75} {MAG*-.58}
POINT 21 {MAG*.78} {MAG*-.09}
POINT 22 {MAG*.75} {MAG*.40}
POINT 23 {MAG*.22} {MAG*.40}
POINT 24 {MAG*.21} {MAG*.78}
POINT 25 {MAG*0.0} {MAG*.80}

```

\$ lines for Tbird

```

LINE 1 STR 1 2
LINE 2 STR 2 3
LINE 3 STR 3 4
LINE 4 STR 4 5
LINE 5 CIRM 5 7 6
LINE 6 STR 7 8
LINE 7 STR 8 9
LINE 8 STR 9 10

```

```

LINE 9 STR 10 11
LINE 10 STR 11 12
LINE 11 STR 12 13
LINE 12 STR 13 14
LINE 13 STR 14 15
LINE 14 STR 15 16
LINE 15 STR 16 17
LINE 16 STR 17 18
LINE 17 STR 18 19
LINE 18 STR 19 20
LINE 19 CIRM 20 22 21
LINE 20 STR 22 23
LINE 21 STR 23 24
LINE 22 STR 24 1 0 7 1.0

$ REGIONS

SIZE {elementSize*MAG}

REGION 1 1 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 *
      -16 -17 -18 -19 -20 -21 -22

SCHEME 0 X
BODY 1
EXIT

```

A command interpreter program, **fsqacs**¹, has been developed to convert FASTQ geometry commands to equivalent ACIS Test Harness commands (outputs an ACIS monitor file). Note, **fsqacs** ignores any meshing information in the FASTQ file since there is currently no means of passing the mesh parameters through the ACIS solid modeler to the CUBIT session. It should be noted that the 2D wireframe geometry can be directly constructed using wires in the ACIS Test Harness; however, there may be instances when it is more convenient to use the command interpreter.

After executing **fsqacs**, the resulting ACIS monitor file may be included in a subsequent ACIS session by simply using the **include** command as illustrated by the following file:

#File: tbird3d.mon

```

include tbird.acs
roll
view scale 200
#draw
cylinder cyll height 1.25 radius 0.5
rotate cyll by 90 about x
#draw
sweep wire f1 by 1.0
#draw
intersect f1 with cyll as tbird3d
#draw
list
save tbird3d to tbird3d.sat

```

Note that the ACIS.mon file demonstrates how 3D solid models may be constructed starting from an initial FASTQ profile followed by typical solid modeling commands (e.g. sweep, intersect) resulting in the desired geometry.

In this example, only the 3D shell of the thunderbird is desired for the finite element model, and thus, the block command is used to specify that only elements on the surface are to be created. The following CUBIT journal file demonstrates current 3D paving capability:

¹. The *fsqacs* users manual is reproduced in Appendix C, “Fsqacs: A FASTQ to ACIS Command Interpreter” on page 163

#File: tbird3d.jou

```

Import Acis 'tbird3d.sat'
#Display
View From 6 3 10
Label Surface
Display
Draw Surface 23
Draw Surface 24

Surface 24 Size 0.03
Surface 24 Scheme Pave
Mesh Surface 24
Draw Surface 24

Block 1 Surface 24
Block 1 Attribute 0.03

```

▼ Assembly Components

Finally, a more practical example of ACIS/CUBIT is demonstrated by meshing an electronics assembly package. Figure B-7 shows a section of the assembly model containing three components: the accelerometer, the timer, and the radar. Also shown is the low density foam encapsulating these components. Note that the foam is of conical shape and the timer and radar units both have draft angles.

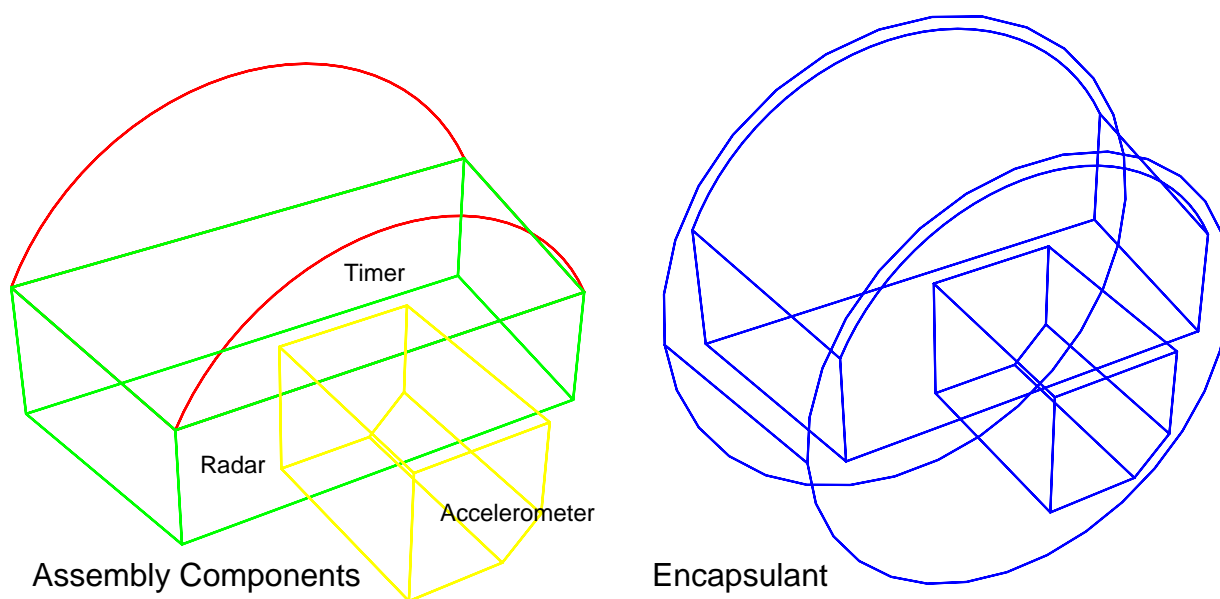


Figure B-7 Components in electronics assembly package.

In this case, the ACIS solid model is constructed on a component by component basis, and the final model called **accelLayer.sat** is generated by grouping the separate ACIS volumes together as one ACIS body. The user may prefer to create the entire solid model in a single ACIS session. However, for demonstration purposes, the model constructed here consists of five ACIS.mon files and one FASTQ input file that is converted to ACIS input using **fsqacs**. A

makefile is used to manage the input and output files and efficiently generate the model. The mesh generated for this assembly is shown in Figure B-8.

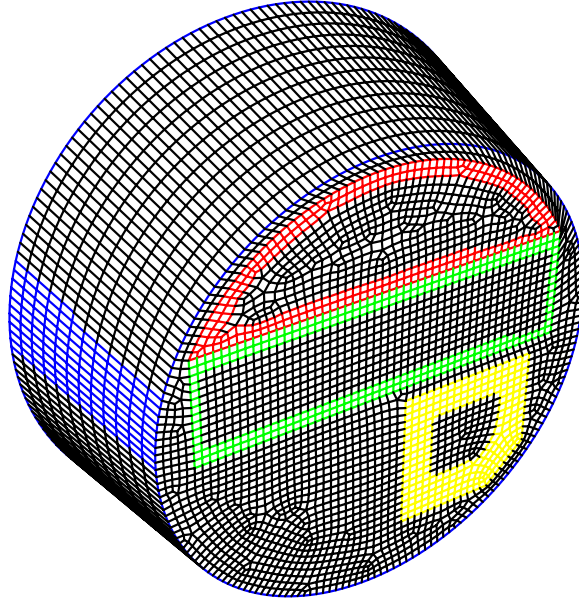


Figure B-8 Generated mesh for the electronics assembly package.

The complete geometric description is given by the following input files.

#File: timer.mon

```
option props on
cylinder cyll height 2.107 radius 2 top 2.362
view from 0 0 1 scale 50

block topBlock width 6 depth 6 height 6
move topBlock 0 3 3
rotate topBlock by -4.41 about x
move topBlock 0 .8976 -1.0535

intersect topBlock with cyll as timer
#draw
save timer to timer.sat
```

#File: radar.mon

```
option props on
cylinder cyll height 2.107 radius 2 top 2.362
block topBlock width 6 depth 6 height 6
move topBlock 0 3 3
rotate topBlock by -4.41 about x
move topBlock 0 .8976 -1.0535

block rightBlock width 6 depth 6 height 6
move rightBlock 3 0 3
rotate rightBlock by 8.734 about y
move rightBlock 0 0 -1.0535
move rightBlock 1.787 0 0
copy rightBlock as leftBlock
```

```

reflect leftBlock along x
unite rightBlock with leftBlock as sliceBlocks
#view from 0 0 1 scale 50
#draw

unite topBlock with sliceBlocks
#draw
subtract sliceBlocks from cyl1 as radar
#draw

block bottomBlock width 6 depth 6 height 6
move bottomBlock 0 -3.125 0

subtract bottomBlock from radar
#draw
save radar to radar.sat

```

#File: accel.mon

```

include accel.acs
view from 0 0 1 scale 50
#draw

sweep wire f1 by 2.107 direction 0 0 1
move f1 0 0 -1.0535
copy f1 as accel
save accel to accel.sat

```

#File: foam.mon

```

option props on
cylinder cyl1 height 2.107 radius 2 top 2.362
view from 0 0 1 scale 50

block rightBlock width 6 depth 6 height 6
move rightBlock 3 0 3
rotate rightBlock by 8.734 about y
move rightBlock 0 0 -1.0535
move rightBlock 1.787 0 0
copy rightBlock as leftBlock
reflect leftBlock along x
unite rightBlock with leftBlock as sliceBlocks
#draw

subtract sliceBlocks from cyl1 as hole
block bottomBlock width 6 depth 6 height 6
move bottomBlock 0 -3.125 0
subtract bottomBlock from hole
#draw hole

retrieve accel.sat as accel
unite accel with hole
#draw hole

cylinder foam height 2.107 radius 2.124 top 2.486
subtract hole from foam
#draw
save foam to foam.sat

```

#File: accelLayer.mon

```

option props on
view from 0 0 1 scale 50

retrieve timer.sat as timer
retrieve radar.sat as radar
retrieve accel.sat as accel
retrieve foam.sat as foam

```

```
#draw

group timer radar accel foam as accelLayer
#draw
save accelLayer to accelLayer.sat
```

ACIS commands worth noting in this example include:

- **option props on** Inserts an edge or “scribe line” along the outer surface of a cylinder. This changes the *periodic*¹ surface into a surface with only one bounding exterior loop of edges. Some CUBIT meshing algorithms require this type of solid model when constructing geometry using cylinders, spheres, or tori.
- **save** Individual components may be saved as separate ACIS solid models.
- **retrieve** Any valid ACIS.sat file may be retrieved and used to perform booleans and/or transformations in an ACIS session.
- **group** Individual components (ACIS bodies) may be grouped together to create a single ACIS.sat file for an assembly.

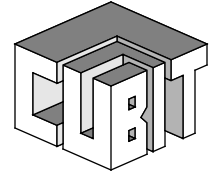
The resulting solid model is meshed in CUBIT using the following commands.

```
#File: accelLayer.jou
journal off
Import Acis 'accelLayer.sat'
Merge All
Display
View From 3 4 -5
Display
# front face of foam encapsulant
Surface 28 Size .07
Surface 28 Scheme Pave
Mesh Surface 28
# front face of accelerometer
Surface 3 Size .07
Surface 3 Scheme Pave
Mesh Surface 3
# front face of radar
Surface 7 Size .07
Surface 7 Scheme Pave
Mesh Surface 7
# front face of timer
Surface 16 Size .07
Surface 16 Scheme Pave
Mesh Surface 16
Display
# foam encapsulant
Volume 4 Interval 12
Volume 4 Scheme Project Source 28 Target 29
Mesh Volume 4
# accelerometer
Volume 3 Interval 12
Volume 3 Scheme Project Source 16 Target 17
Mesh Volume 3
# radar
Volume 2 Interval 12
Volume 2 Scheme Project Source 7 Target 10
Mesh Volume 2
Volume 2 Interval 12
# timer
Volume 1 Scheme Project Source 3 Target 4
```

1. A periodic surface is one which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction.

```
Mesh Volume 1
#Display
Block 1 Volume 1
Block 2 Volume 2
Block 3 Volume 3
Block 4 Volume 4
Export Genesis 'accelLayer.exoII'
```

This example demonstrates that setting the number of intervals for every edge in a 3D solid model can be a very tedious task. When possible, users should use geometry consolidation to reduce the amount of effort involved in performing this step. Additionally, clever use of the body interval command can also significantly reduce time and effort. In this example, all components have the same number of intervals in the z-direction. It is advantageous to set this value for all edges parallel to the z-axis by using the body interval command. Finally, when a mesh is projected from a source surface to a target surface, if one of the surfaces is larger than the other (i.e., if the swept region contains a draft angle), a better quality mesh will usually be generated if the smaller of the two surfaces is used as the source surface.



Appendix C: Fsqacs: A FASTQ to ACIS Command Interpreter

▼ The FASTQ reader imbedded in CUBIT should be used instead of the fsqacs translator. The internal FASTQ reader is accessed by the command `import fastq <filename>`, or by the command line option `-fastq <filename>....163`

▼ Program Execution...163



A text converter program, fsqacs, has been developed to provide a means of generating ACIS solid models from a FASTQ input deck. The program should be considered more as a command interpreter rather than a “true” translator since the output is a set of ACIS Test Harness commands as opposed to an actual solid model.

Note: The FASTQ reader imbedded in CUBIT should be used instead of the **fsqacs** translator. The internal FASTQ reader is accessed by the command **import fastq <filename>**, or by the command line option **-fastq <filename>**.

▼ Description

The **fsqacs** command interpreter is intended to generate a set of ACIS Test Harness commands which will create the same two-dimensional geometry profile as the geometry described in a FASTQ file¹. The output of **fsqacs** is a file which can be input to the ACIS Test Harness to construct a two- or three-dimensional solid model

No mechanism currently exists that allows mesh attributes (for example, interval settings, nodesets, sidesets, and material specifications) to be attached to the ACIS solid model. Therefore, **fsqacs** ignores all mesh information in the FASTQ file. The only FASTQ commands that are converted are those that represent the geometry definition, i.e., POINT, LINE, SIDE, REGION, HOLE, and BODY commands. The mesh interval field in the LINE command and the Element Block ID field in the REGION command are ignored.

▼ Program Execution

1. FASTQ is the two-dimensional mesh generation program previously used by most analysts at Sandia National Laboratories. The fsqacs translator was written to provide a means for analysts to continue work in progress which previously used FASTQ, to try CUBIT on geometries previously meshed using FASTQ and to provide some measure of backward compatibility.

The **fsqacs** program is executed using the following UNIX command:

fsqacs [-aprepro] [-nocover] [-tolerance <val>] in_file.fsq [outputfile]

If no output file is specified, the default is: **in_file.acs**.

The command line options are:

-tolerance <value> Specifies the distance used in ACIS to determine whether two lines intersect. See the following text for more information.

-nocover Specifies that the two-dimensional profile will be subsequently used to generate three-dimensional geometry in ACIS and therefore, the profile should not be covered to generate a surface.

-aprepro Specifies that the FASTQ file should be processed by Aprepro prior to being translated.

Specifying the correct tolerance value is very important, particularly when the model contains circular lines. Due to computational inaccuracies and roundoff, it is possible that connected lines translated from FASTQ to ACIS may not intersect within the default acis tolerance distance of 1.0e-6. If problems are experienced producing a correctly closed “wire” in ACIS, the tolerance value should be increased. A tolerance value of 1.0e-4 has been a good value for many **fsqacs** users.

The **-nocover** option is provided to control whether a closed ACIS wire should be covered or not covered. By default, **fsqacs** assumes that the FASTQ file will be used to create a two-dimensional surface model which must be “covered” in the ACIS Test Harness to generate a surface. If a three-dimensional solid model is desired, the wire should not be “covered” and the **-nocover** option should be specified.

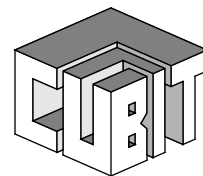
The **-aprepro** option will preprocess the FASTQ input file using Aprepro prior to performing the translation.

The resulting file consisting of ACIS commands must then be processed by executing the ACIS Test Harness to generate the solid model that is imported in CUBIT. See “Importing Geometry” on page 77 and “Examples” on page 147 for more information.

▼ Limitations

Due to the differences in geometry representation between FASTQ and ACIS, **fsqacs** has some limitations:

- Mesh attributes (interval settings, nodesets, sidesets, and material specifications) are not translated.
- Only the STR, CIRC, and CIRM line types are supported. All other line types will be converted to straight lines and a warning message will be printed.
- All points defining a line must be defined prior to encountering the LINE command.
- All lines defining a region must be defined prior to encountering the REGION command.
- A BODY command must be specified.
- The FASTQ file must end with the EXIT command.
- Unless the ACIS Test Harness is reconfigured from the default distributed copy, only 30 edges may be joined to form a region and only 20 regions may be grouped to define a body.



Appendix D: CUBIT Installation

▼ Licensing...	165
▼ Distribution Contents...	166
▼ Installation...	166
▼ HyperHelp Installation...	166

This Appendix contains information about the licensing and redistribution restrictions attached to CUBIT, the distribution contents, and installation instructions. All questions pertaining to obtaining a license for CUBIT should be directed to:

*Marilyn K. Smith
Technology Programs Department
Division 1503, MS-0833
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0833
Fax: (505) 844-9297, Email: mksmith@sandia.gov*

▼ Licensing

CUBIT is distributed in statically linked executable form for each supported platform. Supported platforms include the HP 9000 series running HP-UX¹, Sun SPARCstations running SunOS² and Solaris, and the SGI running IRIX³. Additional platforms will be added as required.

Note: CUBIT installations have use restrictions. THE CUBIT CODE CANNOT BE COPIED TO ANOTHER COMPUTER AND THE NUMBER OF USER SEATS ON EACH COMPUTER OR LAN IS LIMITED. If additional user seats or additional copies of CUBIT are required, you MUST contact us to acquire them.

CUBIT incorporates code modules developed by outside code vendors and our license agreements with them limit the number of user seats at Sandia National Laboratories and limit the number of users who are doing work in conjunction with Sandia National Laboratories.

-
1. HP-UX is a registered trademark of Hewlett-Packard Company.
 2. Sun, SunOS, and Solaris are registered trademarks of Sun Microsystems, Inc.
 3. IRIX is a registered trademark of Silicon Graphics, Inc.

Hence, CUBIT cannot be copied and redistributed without affecting the licensing agreement with the vendors who have proprietary interests in code modules within CUBIT.

Code distributions within Sandia National Laboratories are managed by an informal memorandum. Code distributions outside Sandia National Laboratories are managed by either a Use Notice memorandum or by a formal license agreement depending upon the code recipient. Use Notice and license agreement formats have been developed by the legal department at Sandia National Laboratories to protect the copyrights of code vendors and to protect the commercialization of Sandia National Laboratories copyrights to the CUBIT and SEACAS codes.

▼ Distribution Contents

In addition to the CUBIT executable, a code distribution can include example inputs and a test suite for CUBIT and, depending upon the nature of the request for CUBIT, a code distribution could include certain codes from the Sandia National Laboratories Engineering Analysis Code Access System [14] (SEACAS). Codes in SEACAS which could be used with CUBIT include finite element analysis codes, graphical postprocessing codes, and non-graphical pre- and postprocessing codes. Note that all codes, whether CUBIT or SEACAS codes, run under UNIX¹ operating systems.

Distributions containing other programs in addition to CUBIT will be supplied in tar format. For users who cannot access the tar file through ftp, the tar file will be written to magnetic or CD-ROM media and mailed. Due to possible exposure of the code and subsequent violation of copyrights and export control regulations, no electronic mailing of CUBIT or other codes is permitted.

▼ Installation

CUBIT and supporting CUBIT examples are installed simply by unpacking the tar file and moving the executables to their final directory. Examples and test problems for CUBIT include a README file which provides information needed to run the test problems and examples.

Any SEACAS code distributed with CUBIT will be in source code only. The compilation, linking, and installation of executables is managed by a very complete and extensive installation script. A complete set of installation procedures is provided with the SEACAS codes.

▼ HyperHelp Installation

CUBIT uses an online help system from Bristol Technology called HyperHelp. This online help system allow the viewing of this document and any supporting documentation online. An X Window system is required to run HyperHelp.

1. UNIX is a registered trademark of UNIX Systems Laboratories Inc.

The HyperHelp Viewer files distributed with CUBIT are shown in Table D-1, “HyperHelp Distribution Files,” on page 167.

Table D-1HyperHelp Distribution Files

Filenames	Description
gunzip.hp700.Z gunzip.sun4.Z gunzip.sgi.Z	gunzip utility
hp700R51.tar.gz hp700R52.tar.gz	HyperHelp for HP
sgiR41.tar.gz sgiR42.tar.gz	HyperHelp for SGI IRIX4 and IRIX5
sun4R41.tar.gz sun4R42.tar.gz	HyperHelp for Sun OS 4.1/Solaris 1.1
runtime.tar.gz	Printer Configuration Files for All Platforms

This guide describes how to install your copy of HyperHelp from the HyperHelp installation media. To install HyperHelp 4, you must copy the HyperHelp files from the installation media, set up the HyperHelp environment.

System Requirements

Although HyperHelp4.0 supports many platforms and operating systems, the hardware and software requirement as supplied by the CUBIT distribution are as follows.

Hardware Requirements

• **CPU**

HP 9000 Series 700/800 systems

Silicon Graphics systems

Sun SPARC systems

• **Disk Space**

HyperHelp Viewer: 13MB (includes sample files and printer configuration files)

• **Printer**

Postscript Level 1 and Level 2

PCL Level 4 and Level 5

Software Requirements

• **Operating System**

HP-UX 9.05

IRIX 4.0.5F or IRIX 5.0

SunOS 4.1/Solaris 1.1

SunOS 5.3/Solaris 2.3 (available shortly)

• *Windowing Environment*

X11R5/Motif1.2

X11R4/Motif1.1.4

OpenWindows 3.0

Copying HyperHelp Files

Identify the directory you want to install HyperHelp in, create that directory (if necessary), and **cd** to it.

For example, if you want to install HyperHelp in `/opt/help`, enter the following commands:

```
mkdir /opt/help
```

```
cd /opt/help
```

The HyperHelp installation procedure will create a `hyperhelp` subdirectory in the current working directory.

Uncompress and unzip the HyperHelp files with the following commands:

```
uncompress *.Z
```

```
./gunzip.arch *.gz
```

The *arch* represents your platform architecture (for example, `sun4` or `hp700`).

Unarchive each file that ends with the `.tar` extension as follows:

```
tar xpvf filename.tar
```

The following table shows the HyperHelp installation directory structure:

Directory	Description
<i>install_dir</i> /hyperhelp/bin	Contains the HyperHelp Viewer.
<i>install_dir</i> /hyperhelp/Xp	Contains Xprinter configuration files.
<i>install_dir</i> /hyperhelp/RELEASE	Release information text file.
<i>install_dir</i> /hyperhelp/hoh.hlp	How to Use HyperHelp help file.

Setting Up the HyperHelp Environment

Set the `HHHOME` environment variable to the location of the HyperHelp files.

C shell users: Add the following to your `.cshrc` or `.login` file:

```
setenv HHHOME install_dir/hyperhelp
```

Korn shell or Bourne shell users: Add the following to your `.profile` file:

```
HHHOME=install_dir/hyperhelp;export HHHOME
```

Add `$HHHOME/bin` to your `PATH` environment variable.

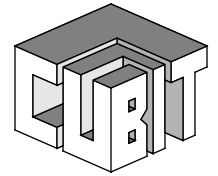
If you have an earlier version of HyperHelp installed on your system, make sure you add `$HHHOME/bin` BEFORE the old HyperHelp path in your `PATH` environment variable.

Activate your HyperHelp X resources with the following command:

cp \$HHHOME/app-defaults /usr/lib/X11/app-defaults/HyperHelp

If you are unable to get access to this directory, you can append the contents of \$HHHOME/app-defaults to \$HOME/.Xdefaults.

Log out and log back in to your system to restart your X server.



Appendix E: Available Colors

Table 6-1 in this Appendix lists the colors available in CUBIT at this time. All color commands require the specification of the color name. The table in this appendix lists the color number (#), color name, and the red, green, and blue components corresponding to each color for reference.

Table 6-1 Available Colors

#	Color Name	Red	Green	Blue
0	black	0.000	0.000	0.000
1	red	1.000	0.000	0.000
2	green	0.000	1.000	0.000
3	yellow	1.000	1.000	0.000
4	blue	0.000	0.000	1.000
5	magenta	1.000	0.000	1.000
6	cyan	0.000	1.000	1.000
7	white	1.000	1.000	1.000
8	grey	0.500	0.500	0.500
9	orange	1.000	0.647	0.000
10	pink	1.000	0.753	0.796
11	brown	0.647	0.165	0.165
12	gold	1.000	0.843	0.000
13	lightblue	0.678	0.847	0.902
14	lightgreen	0.000	0.800	0.000
15	salmon	0.980	0.502	0.447
16	coral	1.000	0.498	0.314
17	purple	0.627	0.125	0.941
18	paleturquoise	0.686	0.933	0.933

Table 6-1 Available Colors

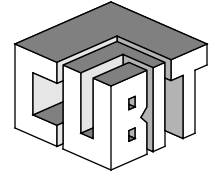
#	Color Name	Red	Green	Blue
19	lightsalmon	1.000	0.627	0.478
20	springgreen	0.000	1.000	0.498
21	slateblue	0.416	0.353	0.804
22	sienna	0.627	0.322	0.176
23	seagreen	0.180	0.545	0.341
24	deepskyblue	0.000	0.749	1.000
25	khaki	0.941	0.902	0.549
26	lightskyblue	0.529	0.808	0.980
27	turquoise	0.251	0.878	0.816
28	greenyellow	0.678	1.000	0.184
29	powderblue	0.690	0.878	0.902
30	mediumturquoise	0.282	0.820	0.800
31	skyblue	0.529	0.808	0.922
32	tomato	1.000	0.388	0.278
33	lightcyan	0.878	1.000	1.000
34	dodgerblue	0.118	0.565	1.000
35	aquamarine	0.498	1.000	0.831
36	lightgoldenrodyellow	0.980	0.980	0.824
37	darkgreen	0.000	0.392	0.000

Table 6-1 Available Colors

#	Color Name	Red	Green	Blue
38	lightcoral	0.941	0.502	0.502
39	mediumslateblue	0.482	0.408	0.933
40	lightseagreen	0.125	0.698	0.667
41	goldenrod	0.855	0.647	0.125
42	indianred	0.804	0.361	0.361
43	mediumspringgreen	0.000	0.980	0.604
44	darkturquoise	0.000	0.808	0.820
45	yellowgreen	0.604	0.804	0.196
46	chocolate	0.824	0.412	0.118
47	steelblue	0.275	0.510	0.706
48	burlywood	0.871	0.722	0.529
49	hotpink	1.000	0.412	0.706
50	saddlebrown	0.545	0.271	0.075
51	violet	0.933	0.510	0.933
52	tan	0.824	0.706	0.549
53	mediumseagreen	0.235	0.702	0.443
54	thistle	0.847	0.749	0.847
55	palegoldenrod	0.933	0.910	0.667
56	firebrick	0.698	0.133	0.133
57	palegreen	0.596	0.984	0.596
58	lightyellow	1.000	1.000	0.878
59	darksalmon	0.914	0.588	0.478

Table 6-1 Available Colors

#	Color Name	Red	Green	Blue
60	orangered	1.000	0.271	0.000
61	palevioletred	0.859	0.439	0.576
62	limegreen	0.196	0.804	0.196
63	mediumblue	0.000	0.000	0.804
64	blueviolet	0.541	0.169	0.886
65	deeppink	1.000	0.078	0.576
66	beige	0.961	0.961	0.863
67	royalblue	0.255	0.412	0.882
68	darkkhaki	0.741	0.718	0.420
69	lawngreen	0.486	0.988	0.000
70	lightgoldenrod	0.933	0.867	0.510
71	plum	0.867	0.627	0.867
72	sandybrown	0.957	0.643	0.376
73	lightslateblue	0.518	0.439	1.000
74	orchid	0.855	0.439	0.839
75	cadetblue	0.373	0.620	0.627
76	peru	0.804	0.522	0.247
77	olivedrab	0.420	0.557	0.137
78	mediumpurple	0.576	0.439	0.859
79	maroon	0.690	0.188	0.376
80	lightpink	1.000	0.714	0.757
81	darkslateblue	0.282	0.239	0.545
82	rosybrown	0.737	0.561	0.561
83	mediumvioletred	0.780	0.082	0.522
84	lightsteelblue	0.690	0.769	0.871
85	mediumaquamarine	0.400	0.804	0.667



Appendix F: CUBIT Application Defaults File

The CUBIT Application Defaults file (CUBIT.ad) file is reproduced here. The use of this file is discussed in “Graphics Customization” on page 21.

```
cubit.defaultFontList: *-helvetica-bold-r-*-*-120-75-75-*-*-iso8859-1
```

```
*webCutShell.background: Grey
```

```
*webCutShell.x: 264
```

```
*webCutShell.y: 113
```

```
cubit*XmTextField*background: LightBlue
```

```
cubit*webCutWindow*XmText*background: AntiqueWhite
```

```
cubit*XmForm*capture_beginBtn.background: Wheat
```

```
cubit*XmForm*capture_endBtn.background: Wheat
```

```
cubit*XmForm*applyBtn.background: Wheat
```

```
cubit*XmForm*helpBtn.background: Wheat
```

```
cubit*XmForm*cancelBtn.background: Wheat
```

```
cubit*XmForm*closeBtn.background: Wheat
```

```
cubit*XmForm*picker_helpBtn.background: Wheat
```

```
cubit*XmForm*picker_applyBtn.background: Wheat
```

```
cubit*XmForm*picker_cancelBtn.background: Wheat
```

```
cubit*XmForm*pickerBtn.background: Wheat
```

```
cubit*consoleShell.background: Grey
```

```
cubit*consoleWindow.x: 16
```

```
cubit*consoleWindow.y: 746
```

```
cubit*consoleWindow.background: Grey
```

```
cubit*XmScrolledWindow.consoleText.background: Grey
```

```
cubit*XmForm.btnRC.background: Grey
```

```
cubit*XmForm.XmRowColumn.btn1.background: Bisque
```

```
cubit*XmForm.XmRowColumn
```

```
.btn2.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn3.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn4.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn5.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn6.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn7.background: Bisque
```

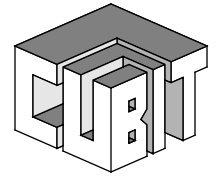
```
cubit*XmForm.XmRowColumn.btn8.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn9.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn10.background: Bisque
```

! Don't recommend changing, but wanted it here for easy changing
cubit*webCutWindow.helpInfo.value: Create a webcut by selecting a body,\n\
supplying a plane for the cut and an optional\n\
vector.\n\
\n\
The plane may be either a face or 3 vertices to\n\
make a face and an optional vector.\n\
\n\
If vector is omitted, default will be 360 degrees.

*cubitmMain.x: 96
*cubitmMain.y: 319
*cubitmMain.title: CUBITM
*cubitmMain.iconName: CUBITM
*cubitmBB.x: 96
*cubitmBB.y: 319
*topLevelShell.x: 595
*topLevelShell.y: 309
*bulletinBoard.x: 595
*bulletinBoard.y: 309
*bulletinBoard.cubitmDA.x: 10
*bulletinBoard.cubitmDA.y: 10
cubit*background: Grey



Appendix G: HyperHelp Viewer

- ▼ Starting the Viewer...177
- ▼ Using Menus and Buttons...177
- ▼ Searching for Specific Information...179
- ▼ Navigating Through Help...181
- ▼ Making Notes on Topics...183
- ▼ Printing Help Topics...184

▼ Starting the Viewer

Although the on-line help system appears to be a seamless part of the parent application, it is actually a separate application that the user must activate. There are three ways to activate the HyperHelp Viewer:

- Choose an option on the parent application's Help menu. The Viewer generally displays a list of topics from which to choose, although the help author can specify any topic to be displayed. Pressing the F1 function key usually performs the same function. Note that the application programmer must include the necessary function call to activate the Viewer.
- Press Shift-F1 to access context-sensitive help, then select an object by clicking on it. If the selected object has been coded by the application developer to provide context-sensitive help, the Viewer immediately displays the appropriate topic. Note that the application programmer must include the necessary function call to activate the Viewer.
- Use the **hyperhelp** command to activate the Viewer in stand-alone mode. This method is very useful for testing the help system before integrating it into your application. For more information about the **hyperhelp** command, see Chapter 9, "Compiling and Testing your HyperHelp System."

▼ Using Menus and Buttons

The HyperHelp Viewer contains four pull-down menus and several special navigation buttons for navigating through the help system in its default configuration. Help authors can use X Window System resources and HyperHelp macros to customize the buttons.

Using HyperHelp Menus

The following table describes the HyperHelp Viewer menus:

Menu Name	Function
File	Open a help file, print help topics, printer setup, and exit.
Edit	Copy or annotate a help topic. Annotated topics display a paper clip icon; click on the clip to view the annotation.
Bookmark	Mark a help topic so you can access it directly from the Bookmark menu.
Help	Explain how to use the HyperHelp Viewer application.

Using HyperHelp Buttons

The following table describes the HyperHelp Viewer buttons:

Button Name	Function
Contents	Display the main index of help topics.
Search	Enter a keyword to search for help.
Back	Display the previous help topic.
History	Display the last 40 help topics viewed.

Using the Keyboard with HyperHelp

Within the HyperHelp Viewer, you can use the mouse to select all button and menu operations, as well as to select hypertext jump terms and defined terms. In addition to the mouse, the HyperHelp Viewer also lets you perform some operations with the keyboard.

Menus

You can open a Viewer menu by holding down the ALT key and pressing the underlined letter in the menu name. The following table lists the ALT key combinations to open Viewer menus:

Key Sequence	Menu
ALT-F	File
ALT-E	Edit
ALT-B	Bookmark
ALT-H	Help

Once a menu is open, you can press the underlined letter in the desired option to choose that option. For example, if the File menu is open, press **x** to exit the Viewer.

You can also use the arrow keys to move around through the open menus. The up and down arrow keys highlight the previous or next items within a menu. For example, if the File menu is open and the Open option is highlighted, you can press the down arrow key to highlight the Print option instead. When the desired menu option is highlighted, you can press the ENTER key to choose that option.

The left and right arrow keys open the menu to the left or right of the currently open menu. For example, if the File menu is open, you can press the right arrow key to open the Edit menu instead.

Buttons

If a Viewer button is highlighted, you can use the left and right arrow keys to highlight the button to the left or right of the currently highlighted button. For example, if the History button is highlighted, you can press the left arrow key to highlight the Search button instead. The HOME key highlights the left-most button. Press the space bar to press the highlighted button.

Scrolling Help Window

If you click your mouse with the cursor pointing inside the scrolling help text window, you can use the PAGE-UP and PAGE-DOWN keys to scroll the window up and down.

▼ Searching for Specific Information

Once you activate the help system, you generally want to find specific information. There are three ways to search for specific information in HyperHelp: looking through a list of help topics, searching for keywords, and searching for specific text.

A keyword is a word or phrase, defined by the help author, that you can use to search for a help topic if you don't know exactly how a topic is named internally in the help system. Instead of looking through the list of help topics, you simply enter a keyword or phrase in the Search dialog; HyperHelp returns a list of topics for which that keyword is defined.

Instead of searching for keywords defined by the help author, text search allows you to search for all occurrences of any text phrase in all help topics. HyperHelp returns a list of topics that contain the phrase, along with the number of times the phrase occurs in that topic.

For example, to find all topics that have information on setting up network printers, you could find all occurrences of *network printers*.

Note: The default HyperHelp configuration does not include full text searching. You must use the **Find()** or **TextSearch()** macro to provide this capability.

To Access Help Topics from the Contents List

1. Press the Contents button. If the help author has provided one, a list of help topics appears in the Viewer window.
2. Point to the desired topic and click the left mouse button. HyperHelp displays the new topic in the Viewer window or a secondary window.

To Search for a Keyword

1. Press the Search button.
2. In the Search dialog, enter the keyword or select a keyword from the list. As you type, HyperHelp scrolls through the list of defined keywords, highlighting the closest match. If there is no defined keyword that matches, try another keyword until you find a match.
3. When your desired keyword is highlighted, press the Find Topics button. HyperHelp displays a list of topic titles for which the keyword is defined.
4. Select the topic that matches the information you are searching for and press the Go To button or double click on the topic title.

To Find Any Text

1. Press the Find button.
2. In the Find dialog, enter the text you want to search for.
3. Select All Text in the Look At options to search the entire help system, including topic titles, for your text. If you only want to search topic titles, select Topic Titles Only.
4. Select Match Case if you want HyperHelp to only find words exactly as you typed them.

For example, to find all instances of *Menu*, but none of *menu*, enter *Menu* in the Find dialog and select Match Case.

5. Press the Search button to find the text.

While HyperHelp is searching for your text, a Search dialog box appears. This box displays the number of topics found so far in the search. If you want to cancel the search, press the Stop Search button.

When the search is complete, HyperHelp displays the Topics Found dialog.

6. To change the sorting order of the topics, press the Sort button. You can sort the results on the order of the topics in the help file, alphabetically, or by the number of hits (instances of the text) found. By default, the results are sorted in the order of the topics in the help file.
7. To display the desired topic, highlight the topic and press the Go To button or double click on the topic title.
8. To display the previous or next topic in the list, press the Previous or Next button.

▼ Navigating Through Help

When you find information about a particular topic, you often want to display help information on related topics. There are several tools for navigating through the help system to find more information:

Navigation Tool	Description
Defined terms	Words or phrases that display a special pop-up help message (usually the definition of the word or phrase) when you press the mouse button. A dotted underline identifies a defined term. When you place your cursor on the defined term, the cursor changes to a pointing finger.
Jump terms	Words or phrases that indicate a cross-reference to a related topic. A solid underline identifies a jump term. When you place your cursor on the jump term, the cursor changes to a pointing finger. When you click the left mouse button, the related topic appears in the main viewer window or a secondary window.
Browse sequences	Series of related topics coded by the help author to be displayed in a specific order. If the current topic is part of a browse sequence, either or both of the Browse buttons is active. Browse sequences apply only if the help author makes the Browse buttons available.
Bookmarks	Let you mark a help topic so that you can quickly retrieve it again in the future using the bookmark name.

To Display a Pop-up Definition

1. Place the cursor anywhere on the defined term. A dotted underline identifies a defined term. The cursor changes to a pointing finger while over the defined term.
2. Press the left mouse button. The definition immediately appears in an overlapping window.
3. To close a pop-up definition, click the left mouse button.

To Jump to a New Topic

1. Place the cursor anywhere on the jump term. A solid underline identifies a defined term. The cursor changes to a pointing finger while over the jump term.
2. Press the left mouse button. The new topic appears in the Viewer window or a secondary window.

To Browse Through a Series of Topics

You may browse through a series of related topics any time the Browse buttons appear on the Viewer button bar. To browse to the next topic in a series, press the Browse>> button. The next topic appears in the Viewer window.

To browse to the previous topic in a series, press the <<Browse button. The previous topic appears in the Viewer window.

To Define a Bookmark

1. While viewing the help topic you wish to mark, choose Define from the Bookmark menu. The Bookmark dialog appears.
2. Press the OK button to accept the topic title as the bookmark reference, or type your own bookmark reference. HyperHelp immediately adds the selected topic to the end of the list of bookmarks.

Note: By default, HyperHelp stores each bookmark in `$HOME/.hh/filename.hlpab`, where `$HOME` is the user's home directory and *filename* is the help file name. If users want to be able to view each other's bookmarks, they must store them in a common directory. To change the default location of bookmarks, each user must change the value of the `HHLOCAL` environment variable by adding the following line to his or her `.profile` (for Bourne shell or Korn shell users):

`HHLOCAL=new_directory_name; export HHLOCAL`

C shell users can change the value of `HHLOCAL` by adding the following entry to their `.cshrc` file:

`setenv HHLOCAL new_directory_name`

If it does not already exist, HyperHelp creates the `.hh` directory in the directory referenced by the `HHLOCAL` environment variable.

To Go To a Bookmark Topic

From the Bookmark menu, choose the topic title you want. HyperHelp immediately displays the associated help topic.

To Delete a Bookmark

1. From the Bookmark menu, choose Define.
2. On the Bookmark dialog, select the bookmark you want to delete and press the Delete button.

▼ Making Notes on Topics

HyperHelp's annotation feature lets you attach a detailed note to a particular topic. For example, if a topic tells you to enter your network address, you may want to annotate the network address for your system.

To Create an Annotation

1. From the Edit menu, choose Annotate.
2. On the Annotate dialog, type your annotation and press the OK button. A paperclip icon appears in the corner of the topic window to indicate that an annotation is associated with the topic.

Note: By default, HyperHelp stores each annotation in `$HOME/.hh/filename.hlpab`, where `$HOME` is the user's home directory and *filename* is the help file name. If users want to be able to view each other's annotations, they must store them in a common directory. To change the default location of annotations, each user must change the value of the `HHLOCAL` environment variable by adding the following line to his or her `.profile` (for Bourne shell or Korn shell users):

`HHLOCAL=new_directory_name; export HHLOCAL`

C shell users can change the value of `HHLOCAL` by adding the following entry to their `.cshrc` file:

`setenv HHLOCAL new_directory_name`

If it does not already exist, HyperHelp creates the `.hh` directory in the directory referenced by the `HHLOCAL` environment variable.

To View an Annotation

1. Point the cursor to the paperclip icon. The cursor changes to a pointing finger while over the icon.
2. Press the left mouse button. The Annotation dialog containing the annotation appears.
3. Press the OK button to close the annotation.

To Delete an Annotation

1. Point the cursor to the paperclip icon. The cursor changes to a pointing finger while over the icon.
2. Press the left mouse button. The Annotation dialog containing the annotation appears.
3. Press the Delete button.

▼ Printing Help Topics

HyperHelp's printing capabilities enable you to print one topic, multiple topics, or all topics in a help file.

To Print the Current Help Topic

1. From the File menu, choose Print. The File Print dialog appears.
2. Select Current Topic and press the OK button. HyperHelp prints the current topic to the selected printer.

Note: If no printer is selected, HyperHelp displays a warning, instructing you to configure a printer first. See "To Configure a Printer."

To Print All Help Topics

1. From the File menu, choose Print. The File Print dialog appears.
2. Select All Topics and press the OK button. HyperHelp prints all topics in the current help file to the selected printer.

Note: If no printer is selected, HyperHelp displays a warning, instructing you to configure a printer first. See "To Configure a Printer."

To Print Selected Help Topics

1. From the File menu, choose Print. The File Print dialog appears.
2. Select Selected Topics and press the OK button. HyperHelp displays a list of all the topics in the current help file in the scrolling list. The list is sorted in the order the topics appear in the file.
3. Select all of the topics you want to print and press the OK button. To skip topics, hold down the CTRL button and deselect topics with the mouse. HyperHelp prints the selected topics to the selected printer.

Note: If no printer is selected, HyperHelp displays a warning, instructing you to configure a printer first. See "To Configure a Printer."

To Configure a Printer

1. From the File menu, choose Printer Setup. The Printer Setup Dialog in Figure 6-1 appears.
2. Set all fields to the desired values. The following table describes all printer setup fields:

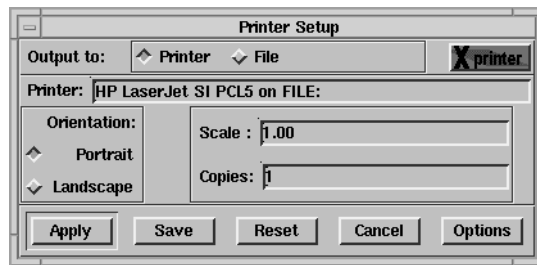
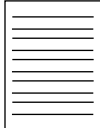
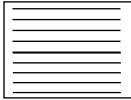
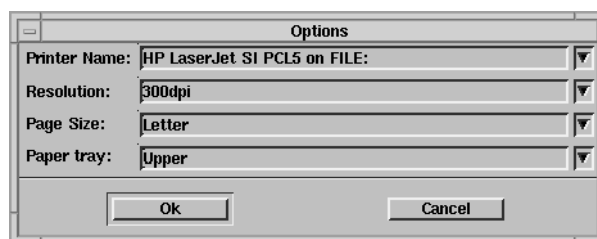


Figure 6-1 Printer Setup Dialog

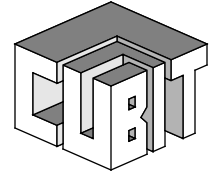
Option	Description
Output to:	Specify whether to send output to a file or to a printer. If you choose Printer, you can send output to any printer configured in your \$HOME/Xpdefaults file. If you choose File, print output is sent to an Encapsulated PostScript or generic PCL file.
Printer:	This field only appears if you select Output to: Printer. It specifies the name of the default printer to send print output to. Press the Options button to specify a different printer.
File Name:	Tihs field only appears if you select Output to: File. Type the name of the print file you wish to create. To pipe print output to a command, type a ! character as the first character and then specify the command to pipe output to. For example, to pipe output to the lp command, enter the following: !lp -d ps .
EPSF	This field only appears if you select Output to: File. Click on this button to display a list of output file types and select the desired type. Available types are EPSF (Encapsulated PostScript), PCL4, and PCL5.
Orientation:	Specify portrait or landscape. <div> <div>  Portrait </div> <div>  Landscape </div> </div>
Scale:	To increase the size of the output, specify a value greater than 1.00. To reduce the size, specify a value less than 1.00. For example, a value of 2.00 would double the size of the output; a value of 0.50 would reduce it by half.
Copies:	Specify the number of copies to print.

- To set additional options, such as selecting a new printer or changing the page size, press the Options button. The Options dialog shown in Figure 6-2 appears.
- Set all options to the desired values. The following table describes all printer options:

**Figure 6-2 Printer Options Dialog**

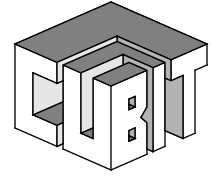
Option	Description
Printer Name:	Use to change the Printer on the Setup Dialog. Press the down arrow button to display a list of configured printers. If the printer you want to print to is not listed, you may configure it by adding it your \$HOME/.Xpdefaults file (see Appendix F, "Using Xprinter with HyperHelp"). You may also use the Output to: File option and pipe output to a command that prints to the desired printer (for example, !rsh texas "lp -d ps" initiates a remote shell on the host texas and executes the lp command).
Resolution:	Specify printer resolution. Values vary among different printers.
Page Size:	Specify paper size. Values vary among different printers.
Paper tray::	Specify tray where paper is located. Values vary among different printers.

5. Press the Save button to make your changes take effect and make them the new default values, or press the Apply button to make your changes take effect without changing the default values.



References

- 1 T. D. Blacker and M. B. Stephenson, 'Paving: a new approach to automated quadrilateral mesh generation', SAND90-0249, Sandia National Laboratories, (1990).
- 2 M. B. Stephenson, S. A. Canann, and T. D. Blacker, 'Plastering: a new approach to automated, 3D hexahedral mesh generation', SAND89-2192, Sandia National Laboratories, (1992).
- 3 G. D. Sjaardema, et. al., *CUBIT Mesh Generation Environment, Volume 2: Developers Manual*, SAND94-1101, Sandia National Laboratories, (1994).
- 4 Spatial Technology, Inc., *ACIS Test Harness Application Guide Version 1.4*, Spatial Technology, Inc., Applied Geometry, Inc., and Three-Space, Ltd., (1992).
- 5 T. D. Blacker, *FASTQ Users Manual Version 1.2*, SAND88-1326, Sandia National Laboratories, (1988).
- 6 L. A. Schoof, *EXODUS II Application Programming Interface*, internal memo, Sandia National Laboratories, (1992).
- 7 W. A. Cook and W. R. Oakes, 'Mapping methods for generating three-dimensional meshes', *Comp. mech. eng.*, **Volume 1**, 67-72 (1982).
- 8 R. E. Jones, *QMESH: A Self-Organizing Mesh Generation Program*, SLA - 73 - 1088, Sandia National Laboratories, (1974).
- 9 R. E. Tipton, 'Grid Optimization by Equipotential Relaxation', unpublished, Lawrence Livermore National Laboratory, (1990).
- 10 A. P. Gilkey and G. D. Sjaardema, *GEN3D: A GENESIS Database 2D to 3D Transformation Program*, SAND89-0485, Sandia National Laboratories, (1989).
- 11 G. D. Sjaardema, *GREPOS: A GENESIS Database Repositioning Program*, SAND90-0566, Sandia National Laboratories, (1990).
- 12 G. D. Sjaardema, *GJOIN: A Program for Merging Two or More GENESIS Databases*, SAND92-2290, Sandia National Laboratories, (1992).
- 13 G. D. Sjaardema, *APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses*, SAND92-2291, Sandia National Laboratories, (1992).
- 14 G. D. Sjaardema, *Overview of the Sandia National Laboratories Engineering Analysis Code Access System*, SAND92-2292, Sandia National Laboratories, (1993).
- 15 S. C. Lovejoy and R. G. Whirley, *DYNA3D Example Problem Manual*, UCRL-MA--105259, University Of California and Lawrence Livermore National Laboratory, (1990).
- 16 Open Software Foundation, Inc., *OSF/Motif™ User's Guide Revision 1.2*, PTR Prentice Hall, Englewood Cliffs, New Jersey, (1993).
- 17 J. M. Osier, *Keeping Track, Managing Messages with GNATS, The GNU Problem Report Management System*, Users manual for GNATS Version 3.2, Cygnus Support, October 1993.
- 18 L. M. Taylor and D. P. Flanagan, *Pronto 3D—A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912, Sandia National Laboratories, (1989).
- 19 S. W. Attaway, unpublished, (1993).



Glossary

B

Body. A body is simply a collection or set of volumes. It differs from volumes only in the fact that booleans are only performed between bodies, not between volumes. The simplest body contains one volume. 68

Brick. A brick is a hexahedral element defined by six connected faces. A brick is owned by the enclosing volume. 90

C

Curve. A curve is a line (not necessarily straight) which is bounded by at least one but not more than two vertices. 67

E

Edge. An edge is defined by a minimum of two nodes. Additional nodes may exist on the edges of higher-order elements. An edge on a curve is owned by that curve, an edge in a surface is owned by that surface, and an edge in a volume is owned by that volume 89

Element Blocks. Element Blocks (also referred to as simply, Blocks) are a logical grouping of elements all having the same basic geometry and number of nodes. 123

F

Face. A face is defined by four connected edges. A face on a surface is owned by that surface, a face in the interior of of a volume is owned by that volume. 90

G

Geometry primitives. Classes of general geometric shapes which are differentiated by basic parameters. CUBIT supports the brick, pyramid, prism, cylinder, torus, frustum, and sphere. 70

H

Hard Point. A vertex which is located in the interior of a surface. It is used to force a node location to that specific geometric location. 67

N

Node A node is a single point in space. A node at a vertex is owned by that geometric vertex, a node on a curve is owned by that curve, a node on the interior of a surface is owned by that surface, and a node in a volume is owned by that volume. 89

Nodeset. Nodesets are a logical grouping of nodes also accessed through a single ID known as the Nodeset ID. 124

P

Periodic Surface. A periodic surface is a surface which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction. 68

S

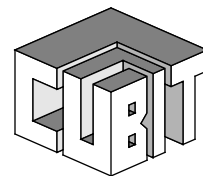
Sideset. Sidesets represent a grouping of element sides and are also referenced using an integer Sideset ID. 124

Surface. A surface in CUBIT is a finite bounded portion of some geometric surface (finite or infinite). A set of surfaces bound the volume in a volume. A surface is bounded by a set of curves. 68

V

Vertex. A vertex occupies a single point in space. A vertex is used to bound a curve and/or to specify a specific location for a node. 67

Volume. Volumes are volumetric regions and are always bounded by one or more surfaces. For practical consideration, volumes will always be bounded by two or more surfaces. 68



Index

Symbols

\$HOME/.cubit 20, 21
.cubit 20, 21
.Xdefaults 21
.Xresources 21

Numerics

2-manifold topology 68

A

ACIS 24, 137, 147
 Import 77
 Test Harness 24, 77, 86, 147, 157, 163, 164
Active
 Window 45
adaptivity 26
airfoil 151
Angle
 Perspective 47, 49, 136
Animation 143
 Pan 136, 139
 Rotate 140
 View 129, 135, 143
 Zoom 136, 145
annotations 183
 creating 183
 deleting 183
 viewing 183
Aprepro 77, 155
-aprepro 164
Arc 77
Aries® ConceptStation 24, 86, 147
aspect ratios 152
Assembly Components 158
At 47, 48, 50, 129, 143
Attribute 124

Block 124, 129

Autocenter 46, 135
Autoclear 46, 51, 135
Axis 47, 135

B

Back button 178
Background Color 44, 53, 131
-batch 20
Bias 97, 133
 Reverse 133
Block 150, 154
 Attribute 124, 129
 Color 53, 131
 Curve 124
 Draw 134
 Element Type 124, 129
 Geometry Color 53
 Geometry Type 129
 Mesh Color 53
 Surface 124
 Visibility 130
 Volume 124
Body 68
 Color 53, 131
 Copy 78, 130
 Decomposition 85, 144
 Draw 51, 134
 Geometry Color 53, 131
 Geometry Visibility 130
 Interval 97, 130, 162
 Interval Size 97, 130
 Label 54, 137
 List 57, 58, 138
 Mesh 138
 Mesh Color 53, 131
 Mesh Visibility 130
 Move 79, 130

- Reflect 81, 130
- Restore 81, 130
- Rotate 80, 130
- Scale 80, 130
- Visibility 53, 130
- Webcut 85, 144
- Bookmark menu 178
- bookmarks 181, 182
 - creating 182
 - deleting 182
 - displaying marked topics 182
- Booleans 82
 - Intersect 82, 137, 149
 - Subtract 82, 141, 148, 149
 - Unite 83, 143
- Border 46, 135
- Boundary Condition 90, 124
 - Contact Surface 124
 - NodeSet 124, 126
 - SideSet 124, 126
- BoundaryLayer 105, 106, 107, 151
 - Curve 131
 - Parameters 131
 - Surface 131
- Box 76
- Box Beam 152
- Brick 70, 131, 132, 148, 149
 - Dialog Box 71
- browse sequences 181, 182
- Button
 - Help 65
- buttons
 - Back 178
 - Contents 178, 179
 - default 178
 - History 178
 - keyboard operations 179
 - Search 178
- By (range) 23

C

- Cellular Topology 68
- Center 46, 47, 135
- Chamfer 77
- Clear 47, 51, 135

- Color 53
 - Background 44, 53, 131
 - Block 53, 131
 - Body 53, 131
 - Geometry 131
 - Mesh 131
 - Dialog Box 54
 - Geometry 131
 - Menu 53
 - Mesh
 - Surface 132
 - Volume 132
 - Node 53, 131
 - NodeSet 53
 - Nodeset 131
 - SideSet 53, 131
 - Surface 53, 132
 - Geometry 132
 - Mesh 132
 - Table 171
 - Volume 53, 132
 - Geometry 132
 - Mesh 132
- Command Line
 - Echo 42, 135
 - Editing 40
 - History 40
 - Interface 39
- configuring
 - printers 184
- Constraints Menu 126
- Contact Surface 124
- Contents button 178, 179
- context-sensitive help 177
- Copy 149
 - Body 78, 130
 - Dialog Box 79
 - Mesh 114, 132
- Create 69
 - Brick 70, 132
 - Cylinder 70, 71, 133
 - Dialog Box 70
 - Frustum 70, 73, 132, 135
 - Prism 70, 72, 132, 140
 - Pyramid 70, 73, 133, 140
 - Sphere 70, 74, 133, 141

- Torus 70, 74, 133, 143
- Window 45
- creating
 - annotations 183
 - bookmarks 182
- Cube with Hole 28, 148
- cubit 19
- CUBIT.ad 22, 175
- CUBIT_HELP_DIR 21
- CUBIT_OPT 21
- cubitb 19, 41
- cubitHelpGUI.hlp 21
- cubitx 19, 21
- Cursor
 - Pan 136, 139
 - Zoom 50, 145
- Curvature
 - Curve Scheme 133
 - Surface Scheme 142
 - Volume Scheme 144
- Curve 67, 125
 - Bias 133, 136, 151
 - Block 124
 - BoundaryLayer 107
 - Curvature 133
 - Delete Mesh 118, 134
 - Draw 51, 134
 - Equal 133
 - Interval 97, 133, 136
 - Interval Size 97, 133, 136
 - Label 54, 137
 - List 57, 58, 138
 - Merge 87
 - Mesh 98, 138
 - NodeSet 126, 139
 - Reverse Bias 97, 133, 136
 - Scheme
 - Bias 97, 133, 136
 - Curvature 133
 - Equal 97, 133
 - SideSet 126, 141
 - Type 133
- Cylinder 70, 71, 132, 133, 148, 149
 - Dialog Box 72

D

- Debug 62
 - List 138
 - Set 141
- debug 20, 62
- Decompose 133
- Decomposition 84, 85, 144
- defined terms 181
- Delete
 - Face 118, 133
 - Mesh 117, 118, 134
 - Window 45
- deleting
 - annotations 183
 - bookmarks 182
- Dialog Box
 - Brick 71
 - Color 54
 - Copy 79
 - Cylinder 72
 - File Selection 42, 43
 - Frustum 73
 - Graphics Draw 51
 - Graphics View 48
 - Hardcopy 55
 - Intersect 82
 - Journal Record/Play 42
 - Merge 87
 - Mesh Delete 118
 - Mode 45
 - Move 80
 - Pause 43
 - Prism 72
 - Pyramid 74
 - Reflect 81
 - Rotate 81
 - Scale 80
 - Sketch 76
 - Sphere 75
 - Subtract 83
 - Torus 75
 - Visibility 52
 - WebCut 84
- DISPLAY 21
- Display 44, 50, 134, 140

Draw 50, 51
 Block 134
 Body 51, 134
 Curve 51, 134
 Edge 51, 134
 Face 51, 134
 Hex 51, 134
 Loop 134
 Menu 51
 Node 51, 134
 NodeSet 51, 134
 SideSet 51, 134
 Skeleton 134
 Surface 51, 134
 Vertex 51, 134
 Volume 51, 134
 Mesh 144

E

Echo 42, 135
 List 138
 Set 141
 Edge
 Draw 51, 134
 Label 54, 137
 Edit menu 178
 Editing
 Mesh 115
 Element Block 25, 123
 Element Type 91, 129
 Block 124
 Encapsulated 136
 Environment Variable
 CUBIT_HELP_DIR 21
 CUBIT_OPT 21
 DISPLAY 21
 HOME 21
 HOOPS_PICTURE 21
 PATH 21
 environment variables
 HHLOCAL 182, 183
 EPS 55, 136
 Equal 97, 133
 Equipotential 116
 Area 116

Error 62
 Example
 Assembly Components 158
 Box Beam 152
 Cube with Hole 28, 148
 Octant of Sphere 149
 Thunderbird 3D Shell 155
 Execution Options
 -batch 20
 -debug 20, 62
 -fastq 20
 -help 20
 -Include 20
 -information 20, 62
 -initfile 20, 21
 -input 20
 -maxjournal 20
 -noinitfile 20
 -nojournal 20, 21, 42
 -solidmodel 20
 -warning 20, 62
 Exit 41, 135
 Exodus 123
 ExodusII 119
 Export
 Genesis 127, 135

F

F1 key 177
 Face
 Delete 118, 133
 Draw 51, 134
 Label 54, 137
 List 57, 58, 138
 False (toggle) 23
 FASTQ 24, 78, 137, 163
 Import 78
 Translator 78
 -fastq 20
 File
 Dialog Box 43
 File menu 178
 File Selection Dialog Box 42
 Filename 23
 Files

- \$HOME/.cubit 20, 21
- .Xdefaults 21
- .Xresources 21
- CUBIT.ad 22, 175
- cubitHelpGUI.hlp 21
- Exodus 123
- ExodusII 119
- Genesis 123, 127, 135
- Menu 41
- Find() 179
- FlatShade 46, 135
- From 47, 48, 50, 135, 143
- Frustum 70, 73, 132, 135
 - Dialog Box 73
- fsqacs 78, 147, 157, 163, 164
 - aprepro 164
 - nocover 164
 - tolerance 164

G

- Genesis 123, 127, 135
 - Export 127
- Geometry
 - Booleans 82
 - Color 131
 - Body 53
 - NodeSet 53
 - SideSet 53
 - Surface 53, 132
 - Volume 53, 132
 - Creation 69
 - Decomposition 84
 - Webcut 84
 - Definition 67
 - Label 54, 137
 - Manipulation 78
 - Menu 70, 75
 - Merge 24, 54, 149, 154, 162
 - Primitives 70
 - Type 129
 - Visibility 52, 130, 135
 - Surface 141
 - Volume 143
- Global 52
- Graphics

- Autocenter 46, 135
- Autoclear 46, 51, 135
- Axis 47, 135
- Border 46, 135
- Center 47, 135
- Clear 47, 51, 135
- Display 44, 50
- Draw 51
- Hardcopy 55
- Line Width 47, 135
- List
 - View 50
- Menu 44, 45, 47, 51
- Mode
 - FlatShade 46, 135
 - HiddenLine 46, 135
 - Painters 46, 135
 - PolygonFill 46, 135
 - SmoothShade 46, 135
 - WireFrame 45, 135
- Mode Type 45
- Pan 136, 139
- Perspective 49, 136
 - Angle 47, 49, 50, 136
- Rotate 49
- Text Size 54
- Window
 - Active 45
 - Create 45
 - Delete 45
- Window Create 45
- Window Size 44, 136
- Zoom 49, 50, 136, 145
 - Cursor 50
 - Reset 50
 - Screen 50

GUI 39

H

- Hard point 67
- Hard Set 93, 95
- Hardcopy 55, 136
 - Dialog Box 55
 - Menu 55
- Hardware Platforms 26

Help 65, 137
 Button 65
 Hyperhelp 65, 137
 Menu 65
 -help 20
 Help menu 177, 178
 Hex
 Draw 51, 134
 Label 54, 137
 List 57, 58, 138
 Weight 116, 144
 Weight Surface 116, 144
 Weighting Function 116
 HHLOCAL 182, 183
 HiddenLine 46, 135
 History button 178
 HOME 21
 HOOPS_PICTURE 21
 HyperHelp
 default buttons 178
 menus 177
 navigation 181
 Viewer 177
 Hyperhelp 65, 137
 hyperhelp command 177

I

Import
 Acis 77, 137
 Fastq 78, 137
 Mesh 103, 119, 137
 -Include 20
 Information 62
 List 138
 Set 141
 -information 20, 62
 -initfile 20, 21
 Initialization File 20
 Initialize
 Video 55, 143
 -input 20
 Intersect 82, 137, 149
 Dialog Box 82
 Interval
 Adjustment 96

Body 97, 130
 Curve 97, 133, 136
 Default 90
 Hard Set 93, 95
 Size
 Body 97, 130
 Curve 97, 133, 136
 Surface 97, 142
 Volume 97, 144
 Specification 91
 Surface 97, 141
 Volume 97, 143

J

Journal
 Dialog Box 42
 List 138
 Pause 43, 140
 Playback 42, 43, 140
 Record 42, 140
 Set 141
 Journal Off 20, 42, 137
 -journalfile
 Execution Options
 -journalfile 20
 jump terms 181

K

keyboard
 using with the Viewer 178
 keyword searching 179, 180

L

Label 53
 All 54, 137
 Body 54, 137
 Curve 54, 137
 Edge 54, 137
 Face 54, 137
 Geometry 54, 137
 Hex 54, 137
 Mesh 54, 137

- Node 54, 137
- Surface 54, 137
- Vertex 54, 137
- Volume 54, 137
- Laplacian 116
- Length-weighted Laplacian 116
- Line 76
- Line Width 47, 135
- List 56
 - Body 57, 58, 138
 - Curve 57, 58, 138
 - Debug 138
 - Echo 138
 - Face 57, 58, 138
 - Hex 57, 58, 138
 - Information 138
 - Journal 138
 - Model 138
 - Node 138
 - Nodes 57, 58
 - Settings 62, 138
 - Surface 57, 58, 138
 - Totals 138
 - Vertex 57, 58, 138
 - View 50, 138, 143
 - Volume 57, 58, 138
 - Warning 138
- List Debug 138
- Loop 110, 134
 - Draw 134

M

- macros
 - Find() 179
 - TextSearch() 179
- makefile 155
- Manifold model 68
- Map 99
 - Scheme
 - Surface 142
 - Volume 108, 144
- maxjournal 20
- Menu
 - Color 53
 - Constraints 126

- Draw 51
- Geometry 70, 75
- Graphics 44, 45, 47, 51
- Graphics Mode Type 45
- Merge 87
- Special 42
- View 47
- Visibility 50
- menus
 - Bookmark 178
 - default 177
 - Edit 178
 - File 178
 - Help 178
 - keyboard operations 178
- Merge 24, 54, 149, 154, 162
 - All 86, 87, 138
 - Curve 87
 - Dialog Box 87
 - General 86
 - Menu 87
 - Only Curves 87
 - Only Surfaces 87
 - Surface 87
 - Vertex 87
- Mesh
 - Body 138
 - Color 131
 - Block 53
 - Body 53
 - NodeSet 53
 - SideSet 53
 - Surface 53, 132
 - Volume 53
 - Copy 114, 132
 - Curve 98, 138
 - Delete 117, 118, 134
 - Dialog Box 118
 - Deletion 117
 - Editing 115
 - GUI 116
 - Import 103, 119, 137
 - Label 54, 137
 - Modify Smooth 115
 - Smooth
 - Modify 115

- Surface 107, 139
- Visibility 52, 130, 139
 - Surface 142
- Volume 107, 114, 139
 - Visibility 144

Messages

- Debug 62
- Error 62
- Information 62
- Warning 62

Mode

- Dialog Box 45

Model

- attributes 25
- List 138

Move

- Body 79, 130
- Dialog Box 80
- NodeSet 117, 139

N

- No (toggle) 23
- nocover 164

Node

- Color 53, 131
- Draw 51, 134
- Label 54, 137
- List 57, 58, 138
- Repositioning 117
- Visibility 52, 139

NodeSet 25, 124, 126

- Color 53, 131
- Curve 126, 139
- Draw 51, 134
- Geometry Color 53
- Mesh Color 53
- Move 117, 154
- Move To 139
- Surface 126, 139
- Vertex 126, 139
- Visibility 52, 139, 141
- Volume 126, 139

- noinitfile 20

- nojournel 20, 21, 42

- Non-manifold topology 68

O

- Octant of Sphere 149

- Off (toggle) 23

- On (toggle) 23

- option props on 161

Output

- PICT 55

- PostScript 55

P

- Painters 46, 135

- Pan 139

- Cursor 136

- Parameter 23

- Optional 23

- PATH 21

- Pause 43, 140

- Dialog Box 43

- Pave 24, 99, 148, 149, 151

- Surface Scheme 142

- Perspective 49, 136

- Angle 47, 49, 50, 136

- PICT 55

- Plaster 26

- Volume Scheme 108, 113, 144

- Playback 42, 43, 140

- Plot 140

- PolygonFill 46, 135

- PostScript 55, 136

- PostScript Begin 26

- PostScript End 55

- Primitives 69

- Arc 77

- Box 76

- Brick 70, 131, 132

- Chamfer 77

- Cylinder 70, 71, 132, 133

- Dialog Box 70

- Frustum 70, 73, 132, 135

- Geometry 70

- Line 76

- Prism 70, 72, 132, 140

- Pyramid 70, 73, 133, 140

- Round 77

Sphere 70, 74, 133, 141
 Torus 70, 74, 133, 143
 printers
 configuring 184
 printing
 all topics 184
 configuring printers 184
 current topic 184
 selected topics 184
 Prism 70, 72, 132, 140
 Dialog Box 72
 PRO/Engineer 24, 77, 86, 147
 Project 149
 Volume Scheme 108, 110, 111, 144
 Pyramid 70, 73, 133, 140
 Dialog Box 74

Q

Quit 41, 135, 140

R

Range 23
 Record 42, 140
 Stop 43, 140
 Reflect
 Body 81, 130
 Dialog Box 81
 Repositioning
 Node 117
 Reset 41, 140
 View 143
 Zoom 50, 145, 212
 Restore
 Body 81, 130
 Reverse Bias 97, 133
 Rotate 49, 140, 149
 Body 80, 130
 Continuous 49
 Dialog Box 81
 Volume Scheme 108, 110, 113, 144
 Round 77

S

Scale
 Body 80, 130
 Dialog Box 80
 Scheme 90
 Bias 97
 Curvature 133, 142
 Designation 108
 Equal 97
 Map 99
 Surface 142
 Volume 108
 Pave 99, 142
 Plaster 108, 113
 Project 108, 110, 111
 Rotate 108, 110, 113
 Sweep 110
 Translate 108, 110, 112
 Triangle 99, 100, 142
 Volume 108
 Curvature 144
 Map 144
 Plaster 144
 Project 144
 Rotate 144
 Translate 144
 Weave 144
 Weave 108, 113
 Screen
 Zoom 145
 Search button 178
 searching
 full text 179, 180
 keyword 179, 180
 Selective 52
 Set
 Debug 141
 Echo 141
 Information 141
 Journal 141
 Warning 141
 Settings
 List 62, 138
 Shift-F1 key 177
 SideSet 25, 124, 126

- Color 53, 131
- Curve 126, 141
- Draw 51, 134
- Geometry Color 53
- Mesh Color 53
- Surface 126, 141
- Visibility 52, 141
- Size
 - Body 130
 - Curve 133, 136
 - Surface 142
 - Volume 144
- Skeleton
 - Draw 134
- Sketch
 - Arc 77
 - Box 76
 - Chamfer 77
 - Dialog Box 75, 76
 - Line 76
 - Round 77
- Smooth
 - Equipotential 115, 116
 - Equipotential Area 116
 - Equipotential Fixed 115
 - Equipotential Generic 116
 - Equipotential Inverse Area 115, 116
 - Equipotential Jacobian 115
 - GUI 116
 - Laplacian 116
 - Length-weighted Laplacian 116
 - Menu 116
 - Method
 - GUI 117
 - Modify Mesh 115
 - Scheme 142, 144
 - Surface 115, 141
 - Volume 116, 141
 - Weight
 - Area 115
 - Inverse Area 115
 - Jacobian 115
- SmoothShade 46, 135
- Snap
 - Video 56, 143
- solidmodel 20
- Source Surface 108, 162
- Special Menu 42
- Sphere 70, 74, 133, 141, 149
 - Dialog Box 75
- stand-alone mode 177
- Step (range) 23
- String 23
- Subtract 82, 141, 148, 149
 - Dialog Box 83
- Surface 68, 125, 142
 - Block 124
 - BoundaryLayer 107
 - Color 53, 132
 - Copy
 - Mesh 114
 - Curvature 142
 - Delete Mesh 118, 134
 - Draw 51, 134
 - Geometry Color 53, 132
 - Interval 97, 141
 - Interval Size 97, 142
 - Label 54, 137
 - List 57, 58, 138
 - Mapping 99
 - Merge 87
 - Mesh 107, 139
 - Visibility 141, 142
 - Mesh Color 53, 132
 - NodeSet 126, 139
 - Scheme
 - Curvature 142
 - Map 99, 142
 - Pave 99, 142
 - Triangle 99, 142
- SideSet 126, 141
- Smooth 115, 141
 - Equipotential 115
 - Equipotential Area 116
 - Equipotential Fixed 115
 - Equipotential Generic 116
 - Equipotential Inverse Area 116
 - Equipotential Jacobian 116
 - Scheme 142
 - Weight
 - Area 115
 - Inverse Area 115

- Jacobian 115
- Source 108
- Target 108
- Visibility 53, 142
- Weight Hexes 116, 144
- Sweep
 - Volume Scheme 110

T

- Target Surface 108
- Test Harness 24
- Text Size 54
- TextSearch() 179
- Through (range) 23
- Thru (range) 23
- Thunderbird 3D Shell 155
- Title 127, 142
- To (range) 23
- Toggle 23
- tolerance 164
- topics
 - printing all 184
 - printing current 184
 - printing selected 184
- Topology
 - 2-manifold 68
 - Cellular 68
 - Non-manifold 68
- Torus 70, 74, 133, 143
 - Dialog Box 75
- Totals
 - List 138
- Transform 78
- Translate 148
 - Volume Scheme 108, 110, 112, 144
- Triangle 99, 100
 - Surface Scheme 142
- True (toggle) 23

U

- Unite 83, 143
- Up 47, 48, 50, 143
- User interface 39

V

- Version 41, 143
- Vertex 67, 143
 - Delete Mesh 118, 134
 - Draw 51, 134
 - Label 54, 137
 - List 57, 58, 138
 - Merge 87
 - NodeSet 126, 139
 - Visibility 52, 143
- Video 55
 - Initialize 55, 143
 - Snap 56, 143
- View
 - At 47, 48, 50, 129, 143
 - Autocenter 46
 - Autoclear 46, 51
 - Border 46
 - Clear 51
 - FlatShade 46
 - From 47, 48, 50, 135, 143
 - HiddenLine 46
 - List 50, 138, 143
 - Menu 47
 - Painters 46
 - Perspective
 - Angle 50
 - PolygonFill 46
 - Reset 143
 - SmoothShade 46
 - Up 47, 48, 50, 143
 - Window Size 44
 - WireFrame 45
- Viewer
 - stand-alone mode 177
 - starting 177
 - using the keyboard with 178
- viewing
 - annotations 183
- Visibility 52
 - Block 130
 - Body 53, 130
 - Body Mesh 130
 - Dialog Box 52
 - Geometry 52, 130, 135

- Volume 143
- Global Geometry Type 52
- Menu 50
- Mesh 52, 139
 - Surface 142
- Mode 52
- Node 52, 139
- NodeSet 52, 139, 141
- SideSet 52, 141
- Surface 53
 - Geometry 141
 - Mesh 142
- Vertex 52, 143
- Volume 53, 144
 - Mesh 144
- Volume 68, 125
 - Block 124
 - Color 53, 132
 - Copy
 - Mesh 114
 - Delete Mesh 118, 134
 - Draw 51, 134
 - Geometry
 - Color 132
 - Visibility 143
 - Geometry Color 53
 - Interval 97, 143
 - Size 144
 - Interval Size 97
 - Label 54, 137
 - List 57, 58, 138
 - Map 108
 - Mesh 114, 139
 - Color 132
 - Draw 144
 - Visibility 144
 - Mesh Color 53
 - Meshing 107
 - NodeSet 126, 139
 - Scheme 108
 - Curvature 144
 - Map 108, 144
 - Plaster 108, 113, 144
 - Project 108, 110, 111, 144
 - Rotate 108, 110, 113, 144
 - Sweep 110

- Translate 108, 110, 112, 144
- Weave 108, 113, 144
- Smooth 116, 141
 - Equipotential 116
 - Equipotential Fixed 116
 - Laplacian 116
 - Scheme 144
- Visibility 53, 144

W

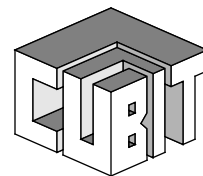
- Warning 62
 - List 138
 - Set 141
- warning 20, 62
- Weave
 - Volume Scheme 108, 113, 144
- Web Cutting 84
- Webcut
 - Body 85, 144
 - Dialog Box 84
- Weight 115
- Weight Hexes
 - Surface 116, 144
- Weighting Function
 - Hex 116
- Window
 - Active 45
 - Create 45
 - Delete 45
- Window Create 45
- Window Size 44, 136
- WireFrame 45, 135

Y

- Yes (toggle) 23

Z

- Zoom 49, 50, 136, 145
 - Cursor 50, 145
 - Reset 50, 145
 - Screen 50, 145



Appendix H: ERRATA — May 26, 1994

The page number of the form XX.Y indicates that the change is to page XX of the manual and appears approximately Y0% down the page. For example, Page 12.9 indicates that the change is on Page 12 of the manual, 90% down from the top of the page. The errata are for “Document Version 5/23/94”

Chapter 1

- Page 12.9: New command line option: -noecho controls echoing of commands to the terminal or GUI console window. Echoing is on by default.
- Page 14.5:
cubit.graphics.geometry: 700x700+445+0
- Page 21.1: A string parameter is a literal character string contained within single *or* double quotes.

Chapter 2

- Page 46: Graphics Mode Dialog Box: Ordering of modes is: Wireframe, Hiddenline, Polygonfill, Painters, Flatshade, Smoothshade. Text for button in middle of window is “Open View Dialog Box”, Axis Button should not be “grayed out”

Chapter 3

- Page 54, Figure 3-2: Mesh and Graphics Menu items swapped.
- Page 55, Figure 3-4: Executing Command: *Graphics* WindowSize 700 700.
- Page 60, Figure 3-8: Graphics Mode Dialog Box: Ordering of modes is: Wireframe, Hiddenline, Polygonfill, Painters, Flatshade, Smoothshade. Axis Button should not be “grayed out”
- Page 62, Figure 3-10: **VIEWPORT** button is future technology. Functionality not implemented.

Chapter 5

- Page 105, Figure 5-8: “First Layer Depth” should be “Sublayer depth”, “Second Layer Depth” should be “First Layer Depth.”

Appendix A (Command Index)

- Page 124. Block Commands: **BlockId** keyword is replaced by **Block**
- Page 124. Block Element Type command:
—**ElementType** is replaced by **Element Type** (two words).

- Page 129. Graphics Line Width command:
— **LineWidth** is replaced by **Line Width** (two words).
- Page 129. Graphics Mode command:
— **FlatShade** replaced by **Flat Shade**
— **HiddenLine** replaced by **Hidden Line**
— **PolygonFill** replaced by **Polygon Fill**
— **SmoothShade** replaced by **Smooth Shade**
- Page 129. Graphics Window Size command:
— **WindowSize** replaced by **Window Size**
- Page 129. New Command:

Graphics Zoom Screen

page 50

[Graphics] Zoom Screen <Scale_Factor>

- Page 130. Journal Off command. Journal can now be turned off and on. New syntax:

Journal

page 42

Journal {on | off}

- Page 132. Playback command: <journal_filename> should be surrounded by single quotes.

Appendix F

- Page 160.5: Added line to file:
cubit.graphics.geometry: 500x500+0+0